Technical Report 1170

Computational Structure of GPSG Models: Revised generalized phrase structure grammar

Eric Sven Ristad

MIT Artificial Intelligence Laboratory

This blank page was inserted to preserve pagination.

MASSACHUSETTS INSTITUTE OF TECHNOLOGY
ARTIFICIAL INTELLIGENCE LABORATORY

A.I. T.R. No. 1170

September, 1987
Revised September, 1989

COMPUTATIONAL STRUCTURE OF GPSG MODELS:
Revised generalized phrase structure grammar

Eric Sven Ristad

A central goal of mathematical linguistics is to precisely determine the power of a linguistic theory. Traditionally, formal language theory (the Chomsky hierarchy) and its generative power analyses have translated this question into the narrower question of how unrestricted the rule format of a theory is. Modern computational complexity theory offers another, more useful, translation: how much of what computational resources does a theory consume? Complexity theory also offers a new perspective on descriptive adequacy. In a descriptively adequate linguistic theory, the structural descriptions *and* computational power of the theory match those of an ideal speaker-hearer.

The primary goal of this essay is to demonstrate how considerations from computational complexity theory can inform grammatical theorizing. To this end, the essay revises generalized phrase structure grammar (GPSG) linguistic theory so that its computational power more closely matches the limited computational ability of an ideal speaker–hearer. A second goal is to provide a theoretical framework within which to better understand the wide range of GPSG models that have appeared in the theoretical and computational linguistics literature, embodied in formal definitions as well as in implemented computer programs.

The essay begins with an outline and intuitive complexity analysis of the GPSG formal system of Gazdar, Klein, Pullum, and Sag (1985). Subsequently, revisions to the formal system are motivated by complexity and generative concerns. The revised system is presented along with a detailed account of topicalization, expletive pronouns, and parasitic gaps. An extensive RGPSG for English is included in an appendix. This work falls within the GPSG approach to linguistics. Revised GPSG is, however, less opaque, more tractable, and more linguistically constrained than standard GPSG theory: GPSG Recognition is EXP-POLY time hard, while RGPSG Recognition is NP-complete.

Acknowledgments: I wish to thank Ed Barton, Robert Berwick and Richard Larson for improvements, carifications, helpful discussion, and the computation tree figures; Sandiway Fong for simplifications; and Geoff Pullum for his critical questions and patient help with GPSG theory. The length and quality of this essay have both benefited from the concerns of five anonymous reviewers.

The author is supported by a graduate fellowship from the IBM Corporation. This report describes research done in part at the Artificial Intelligence Laboratory of the Massachusetts Institute of Technology. Support for the Laboratory's artificial intelligence research has been provided in part by a grant from the Kapor Family Foundation, in part by NSF Grant DCR-85552543 under a Presidential Young Investigator Award to Professor Robert C. Berwick, and in part by the Advanced Research Projects Agency of the Department of Defense under Office of Naval Research contract N00014-85-K-0124. Earlier versions of portions of this essay in its infancy were presented at the 1986 and 1987 Association of Computational Linguistics Conferences, at Columbia and Stanford Universities. The essay will appear in Vol.13 of *Linguistics and Philosophy.* This report is a completely rewritten version of a thesis submitted to the MIT Department of Electrical Engineering and Computer Science in June 1986, in partial fullfillment of the requirements for the degree of Master of Science.

1 Introduction and Motivation

A linguistic theory specifies a computational process that assigns structural descriptions to utterances. This process requires certain computational resources, such as time or space. In a descriptively adequate linguistic theory, the computational resources used by the theory match those used by an ideal speaker-hearer. In this essay, I explain exactly how computational complexity analysis can be used to revise generalized phrase structure grammar (GPSG) so that its computational power more closely corresponds to the limited ability of an ideal speaker–hearer. This work falls within the GPSG approach to linguistics, as presented in Gazdar, Klein, Pullum, and Sag (1985), GKPS hereafter. Revised GPSG is, however, less opaque, more tractable, and more linguistically constrained than GPSG: GPSG Recognition is EXP-POLY time hard, while RGPSG Recognition is NP-complete.

Computational complexity theory measures the intrinsic lower-bound difficulty of obtaining the solution to a problem no matter how the solution is obtained. It classifies problems according to the amount of computational resources (for example, time, space, electricity) needed to solve them on some abstract machine model, typically a deterministic Turing machine. Complexity classifications are invariant across a wide range of primitive machine models, all choices of representation, algorithm, and actual implementation, and even the resource measure itself. For linguists, complexity analysis provides a new answer to the central question of formal linguistics: how powerful is a linguistic theory? For computational linguists, it provides a precise implementation-independent cost measure necessary for informed parser engineering.

The bulk of this essay is devoted to informally identifying what computational resources are used by GPSG theory, and determining whether they are linguistically necessary. GPSG contains five formal devices, each of which is used to model some linguistic phenomenon or ability, and each of which requires certain computational resources. I identify those aspects of each device that cause intractability and then restrict the computational power of each device to more closely match the (inherent) complexity of the phenomenon or ability it models. This method reveals the tension between descriptive adequacy and explanatory power that, when precisely focused by complexity analysis, I find fascinating. The remainder of the essay presents the new formal system and exercises it in the domain of topicalization, ex-

pletive pronouns, and parasitic gaps. The conclusion places this work in perspective in mathematical linguistics.

In my opinion, the primary value of this work lies in the result (*revised* GPSG, or RGPSG) as well as in its use of complexity analysis to understand and improve a major linguistic theory. RGPSG is of value both to linguists and computational linguists because it is more tractable and easier to understand, use, and implement. It can be efficiently implemented and appears to have better empirical coverage that its GPSG ancestor, in addition to fixing some errors in GKPS. It would be informative for the reader to compare this work to other "revised GPSGs," such as the head-driven phrase structure grammar of Pollard (1984) and the unification categorial grammar of Zeevat, Klein, and Calder (1987).

2 The GPSG Formal System

A GPSG is a formal model of linguistic competence. As a linguistic model, it must encode linguistically significant relations, such as domination and predication, and it must constrain their distribution. And, as it is formal, the model must be perfectly explicit. This section outlines the GPSG formal system, as presented in Gazdar, Klein, Pullum, and Sag (1985), GKPS hereafter, and explains how abstract linguistic relations are formally encoded in GPSG, and how these relations are formally constrained.

2.1 Overview of GPSG Formalisms

From the perspective of classic formal language theory, a GPSG may be thought of as a grammar for generating a context-free grammar. The generation process begins with immediate dominance (ID) rules, which are context-free productions with unordered right-hand sides. An important feature of ID rules is that nonterminals in the rules are not atomic symbols (for example, NP). Rather, GPSG nonterminals are sets of [*feature feature-value*] pairs. For example, [N +] is a [*feature feature-value*] pair, and the set {[N +], [V -], [BAR 2]} is the GPSG representation of a noun phrase.[1] Next, metarules apply to the ID rules, resulting in an enlarged set of ID rules. Metarules have fixed input and output patterns containing a distinguished multiset variable W in addition to constants. If an ID rule matches the input pattern under some specialization of the variable W, then the metarule generates an ID rule corresponding to the metarule's output pattern under the same specialization of W. For example, the passive metarule

$$\begin{array}{c} VP \rightarrow W, NP \\ \Downarrow \\ VP[\texttt{PAS}] \rightarrow W, (PP[\texttt{by}]) \end{array} \tag{1}$$

says that "for every ID rule in the grammar which permits a VP to dominate an NP and some other material, there is also a rule in the grammar which permits the passive category VP[PAS] to dominate just the other material from the original rule, together (optionally) with a PP[by]" (GKPS:59).

[1] Although syntactic categories in GPSG are not atomic symbols, they are traditionally abbreviated (up to ambiguity) by atomic symbols such as "*NP*" (as explained below), which has confused some readers.

Below we use the *finite closure* problem to determine the cost of applying metarules in this manner. Principles of universal feature instantiation (UFI) apply to the resulting enlarged set of ID rules, defining a still larger set of phrase structure trees of depth one (local trees). One principle of UFI is the head feature convention, which ensures that phrases are projected from lexical heads. Informally, the head feature convention is GPSG's X'-theory. We will use the *category projection* problem to determine, in part, the cost of mapping ID rules to local trees. Finally, linear precedence statements are applied to the instantiated local trees. LP statements order the unordered daughters in the instantiated local trees. The ultimate result, therefore, is a set of ordered local trees, and these are equivalent to the context-free productions in a context-free grammar. The resulting context-free grammar derives the language of the GPSG.

From the perspective of a linguist, GPSG theory contains five language-particular components: immediate dominance rules, metarules, linear precedence constraints, feature co-occurrence restrictions (FCRs), and feature specification defaults (FSDs).[2] GPSG theory also provides four language-universal components: a theory of syntactic features, principles of universal feature instantiation, principles of semantic interpretation, and formal relationships among various components of the grammar. In this section, I provide a brief and linguistically motivated overview of the theory.

2.2 Syntactic categories

In current GPSG theory, syntactic categories (nonterminals) encode abstract linguistic relations and properties as feature-value pairs. Categories encode subcategorization, agreement, unbounded dependency, predication, and other syntactically significant relations. If a relation is true of two categories in a phrase structure tree, then the relation will be encoded in every category on the unique path between the two categories. For example, the feature SLASH encodes the gap–filler (unbounded dependency) relation. Therefore, every category on the path from a gap to its filler will have a SLASH feature whose value is the category of the filler. Similarly, categories that are assigned nominative case by a sister category in a local tree will have a CASE feature whose value is NOM.

[2]The following description of the GPSG formal system is taken with substantive modifications from Barton, Berwick, and Ristad (1987).

More formally, GPSG categories are partial functions that map features to atomic feature values or to syntactic categories. Categories may also be thought of as sets of *feature specifications*. A feature specification is a pair [*feature feature-value*] where *feature* is an atomic symbol and *feature-value* is either an atomic symbol or a syntactic category. Thus, [N +] indicates that the atomic-valued "nominal" feature has the "+" value, while [SLASH {[BAR 2]}] indicates that the "slash" feature has the category {[BAR 2]} as its value. In the GPSG system, the feature set {[N +], [V -], [BAR 2]} represents a noun phrase, {[N -], [V +], [BAR 2], [SUBJ -]} is a verb phrase, and {[N -], [V +], [BAR 2], [SUBJ +]} is a verb phrase with a subject, or a clause. Some features are morphologically realized; for example, in the GKPS grammar for English, a category bearing the feature specification [PFORM with] is a prepositional category headed by the preposition *with*.

I adopt the abbreviatory conventions found in the GPSG literature: syntactic categories may be abbreviated up to ambiguity. Thus, a noun phrase containing the additional feature specifications [CASE NOM] and [POSS +] might written *NP*[CASE NOM,POSS +] or even as *NP*[NOM,+POSS] because the atomic feature-value NOM may only be associated with the CASE feature. The category-valued SLASH feature is abbreviated with a trailing slash ('/') character: *VP*[VFORM PAS,SLASH *NP*] is usually written *VP*[PAS]/*NP*. A numerical value appearing inside square brackets ([32], for example) denotes a SUBCAT value, while a numerical value that precedes a set of square brackets is a BAR value. For example, the category *V0*[2] abbreviates the category {[N -], [V +], [BAR 0], [SUBCAT 2]}.

The set K of syntactic categories is specified inductively by listing a set *Feat* of features, a set Atom of atomic-valued features, a set A of atomic feature values, a function ρ that defines the range of each atomic-valued feature, and a set R of restrictive predicates on categories (feature co-occurrence restrictions). The *category-valued features* in (*Feat* – Atom) allow categories to be freely contained within other categories, subject to FCRs (below) and the restrictive principle of *finite feature closure*, which prevents a category-valued feature f from taking categories in which f already appears. That is, the feature specification [f C] is legal only "if f is not in the domain of C, or in the domain of any C' contained in C, at any level of embedding" (GKPS:36).

A category C_1 *extends* a category C_2 (written $C_1 \sqsupseteq C_2$) if and only if two conditions hold. For every *atomic* feature specification f in C_2,

it must be true that $C_1(f) = C_2(f)$, and for every *category-valued* feature specification f in C_2, it must be true that $C_1(f) \sqsupseteq C_2(f)$. For example, $\{$[N +], [V -], [BAR 2], [POSS +]$\} \sqsupseteq NP$, and $VP \not\sqsupseteq S$ because $VP($[SUBJ]$) \neq S($[SUBJ]$)$.

2.3 Marking Conventions

As one might imagine, not every set of feature specifications that satisfies finite feature closure is a possible syntactic category. For example, there are no passive prepositional phrases, and a noun phrase cannot bear the [PFORM with] specification, which is reserved for prepositional categories. These constraints are expressed through *feature co-occurrence restrictions* (FCRs) and *feature specification defaults* (FSDs), which are marking conventions used in the GPSG system both to express language-particular facts and to restrict the overgeneration of other formal devices (both metarule and feature closure). FCRs and FSDs are restrictive predicates on categories, constructed by Boolean combination of feature specifications. All legal categories must unconditionally satisfy all FCRs. All categories must also satisfy all FSDs, if it is possible to do so without violating an FCR or a principle of universal feature instantiation. For example,

$$\textit{FCR 1: } \text{[INV +]} \supset (\text{[AUX +]} \wedge \text{[VFORM FIN]})$$

requires any category that bears the [INV +] feature specification to also bear the specifications [AUX +] and [VFORM FIN].

2.4 Immediate Dominance/Linear Precedence

GPSG's immediate dominance/linear precedence format factors out two independent relations that compose phrase structure. An ID rule is a context-free production

$$C_0 \rightarrow C_1, C_2, \ldots, C_k$$

whose left-hand side (LHS) is the mother category and whose right-hand side (RHS) is an unordered multiset of daughter categories, some of which may be designated as *head daughters.* The LHS *immediately dominates* the unordered RHS in a tree of depth one (a *local tree*).

An LP statement is a pair of category predicates

$$P_1 \prec P_2$$

that requires a category C_1 in a local tree to precede it's sister C_2 if C_1 satisfies P_1 and C_2 satisfies P_2. A predicate is a Boolean combination ($\&, \vee, \neg$) of truth-values and feature specifications such that if a category C bears or extends a given feature specification, that feature specification is true of C, else false. For example, the LP statement

$$\texttt{[SUBCAT]} \prec \neg \texttt{[SUBCAT]}$$

requires categories bearing a **SUBCAT** specification to precede categories unspecified for **SUBCAT** (that is, lexical categories must precede nonlexical categories). This LP statement requires lexical heads to precede their complements, and thereby represents the setting of the "head" parameter for English.

The primary advantage of the ID/LP format stems from its partial decoupling of two independent linguistic relations (see McCawley 1982 for arguments that these relations are in fact independent): by decoupling the two relations, GPSG can express the head parameter and capture some free-word order facts.

2.5 Metarules

Metarules are lexical redundancy rules. Formally, they are functions that take *lexical ID rules*—ID rules with a lexical head—to sets of lexical ID rules. Metarules have a fixed input ID rule pattern containing a mother category, at most one daughter category, and a distinguished multiset variable W. W ranges over multisets of daughter categories. If an ID rule matches the input pattern under some extension of the two pattern categories and some specialization of the variable W, then the metarule generates an ID rule corresponding to the metarule's fixed ID rule output pattern under the same extension of pattern categories and same specialization of W. See the GKPS passive metarule above. The GKPS grammar for English includes metarules for subject-aux inversion, extraposition, and transitivity alternations.

The complete set of ID rules in a GPSG is the maximal set that can be arrived at by taking each metarule and applying it to the set of rules that did not themselves arise from the application of that metarule. This

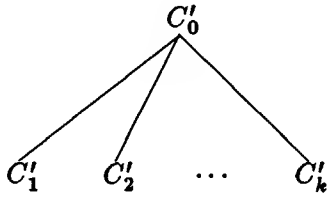
maximal set is called the *finite closure* $FC(M, R)$ of a set R of lexical ID rules under a set M of metarules.

2.6 Local trees

The ID rules obtained by taking the finite closure of the metarules on the ID rules are *projected* to local phrase structure trees. Abstractly, this process establishes the connection between those relations encoded in ID rules (for example, domination, subcategorization, case, modification, and predication) and the nonlocal linguistic relations (for example, gap–filler, agreement, and wh-element scope). Local trees are projected from ID rules by mapping the categories in a rule into legal extensions of those categories in the projected local tree.

$$C_0 \rightarrow C_1, C_2, \ldots, C_k$$

projects to the local tree



where for all i from 0 to k, C'_i extends C_i. Because the RHS of an ID rule is unordered, the C'_i could appear in any order (subject to linear precedence constraints).

Principles of *universal feature instantiation* (UFI) constrain this projection by requiring categories in a local tree to agree in certain feature specifications when it is possible for them to do so. For example, the head feature convention (HFC) requires the mother to agree with all head features that the head daughters agree on, if agreement is possible. The HFC expresses X'-theory in part, requiring a phrase to be the projection of its head. It also plays a central role in the GPSG account of coordination phenomena, requiring the conjuncts in a coordinate structure to all participate in the same linguistic relations with the rest of the sentence. The two other principles of UFI are the *control agreement principle* and the *foot feature principle*. The control agreement principle represents the GPSG theory of predicate–argument relations. Informally, it requires predicates to agree with their

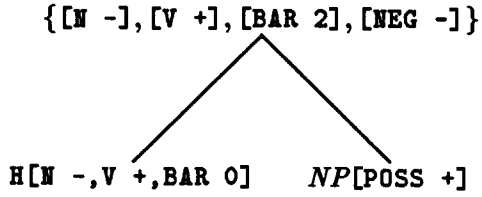
arguments (for example, verb phrases must agree with their subject NPs in English). The foot feature principle provides a partial account of gap–filler relations in the GPSG system, including parasitic gaps and the binding facts of reflexive and reciprocal pronouns; it plays a role strikingly similar to that of Pesetsky's (1982) path theory and Chomsky's (1986) binding and chain theories.[3] The foot feature principle requires foot features instantiated on the mother to be instantiated on at least one of the daughters, and vice versa. Thus the FFP ensures that certain syntactic information is not lost. "Exceptional" feature specifications are those feature specifications in an ID rule that should agree by virtue of a principle of UFI, but are unable to without changing a feature specification inherited from the ID rule.

Local trees are further constrained by FSDs, FCRs, and LP statements. Finally, local trees are assembled to form phrase structure trees, which are terminated by lexical elements.

Let the features {N,V,BAR} and {NEG,POSS} be head features and non-head features, respectively. Let the symbol H mark head daughters. Then the ID rule

$$\{[\mathtt{N}\ -],[\mathtt{V}\ +],[\mathtt{BAR}\ 2]\} \rightarrow \mathtt{H}[\mathtt{BAR}\ 0], NP$$

can project to this local tree:



In this example, BAR is considered an exceptional feature specification because the mother's BAR value (2) conflicts with the head daughter's BAR value (0), and it is impossible to resolve the conflict without changing an existing feature specification.

[3]The possibility of expressing the control agreement and foot feature principles as local constraints on nonlocal relations falls out from the central role of c-command, or equivalently unambiguous paths, in binding theory. Similarly, the possibility of encoding multiple gap–filler relations in one feature specification of one category, as in the GPSG analysis of parasitic gaps, corresponds to the "no crossing" constraint of path theory. Pesetsky (1982:556) compares the predictions of path theory and principles of UFI when the two diverge in cases of double extraction (for example, *a problem that$_i$ I know who$_j$ to* [$_\alpha$ *talk to* e_j *about* e_i]) from coordinate structures. He concludes that "the apparent simplicity of the slash category solution fades when more complex cases are considered."

3 Classifications of Complexity Theory

This section introduces the powerful tool of modern computational complexity analysis in order to apply it to GPSG theory in the next section. Recall that computational complexity theory measures the intrinsic lower-bound difficulty of obtaining the solution to a problem no matter how the solution is obtained. It classifies problems according to the amount of computational resources (in our case, time and space) needed to solve them on a given abstract machine (for example, a deterministic Turing machine).

This paper refers to four complexity classes: $\mathcal{P}$, $\mathcal{NP}$, PSPACE, and EXP-POLY. Below I provide an intuitive geometric characterization of these four classes through their equivalence class representatives (computation trees). The classes are defined algebraically as follows.

3.1 Four Important Complexity Classes

$\mathcal{P}$ is the natural and important class of problems solvable in deterministic $\mathcal{P}$olynomial time, that is, on a deterministic Turing machine in time n^j for some integer j, where n denotes the size of the problem to be solved.[4] $\mathcal{P}$ is considered to be the class of problems that can be solved efficiently. For example, sorting takes $n \cdot \log n$ time in the worst case using a variety of algorithms, and therefore is efficiently solvable.

$\mathcal{NP}$ is the class of all problems solvable in $\mathcal{N}$ondeterministic $\mathcal{P}$olynomial time. Informally, a problem is in $\mathcal{NP}$ if one can guess an answer to the problem and then verify its correctness, all in polynomial time. For example, the problem of deciding whether a whole number i is composite is in $\mathcal{NP}$ because it can be solved by quickly guessing a pair of potential divisors, each less than $\lceil\sqrt{i}\rceil$, and then quickly checking if their product equals i.

PSPACE is the class of problems solvable in deterministic polynomial *space*. PSPACE contains $\mathcal{NP}$ because polynomial space allows us to simulate an entire $\mathcal{NP}$ computation, but it is not known if the inclusion is proper. Intuitively, PSPACE is the class of combinatorial two-person games: it includes the problems of winning generalized versions of Checkers, Go, and Parker Brothers' Instant Insanity(TM). Many problems in formal language

[4] Problems must be encoded in a "reasonable" way for a size measure to make sense; for discussion, see Garey and Johnson (1979).

theory are known to be PSPACE-complete, such as context-sensitive language recognition and finite state automaton inequivalence and intersection.

Finally, *EXP-POLY* is the class of problems solvable in deterministic time $O(c^{f(n)})$ for any constant c and polynomial $f(n)$ in n. This class includes PSPACE, and all exponential time problems, and so includes problems that are *provably* intractable. No natural problems are known to be EXP-POLY-complete, although the universal recognition problem for GPSGs may be.

We say a problem T is *$\mathcal{C}$-hard* (with respect to polynomial time reductions) if T is at least as hard computationally as any problem in the complexity class $\mathcal{C}$: if we had a subroutine that solved T in polynomial time, then we could write a program to solve any problem in $\mathcal{C}$ in polynomial time on a deterministic Turing machine (essentially by efficiently transforming the problem in $\mathcal{C}$ to T and then solving T with the fast subroutine). Note that T need not be in $\mathcal{C}$ to be $\mathcal{C}$-hard. A problem is *$\mathcal{C}$-complete* if it is both $\mathcal{C}$-hard and included in $\mathcal{C}$.

NP-complete problems can be solved only by methods too slow for even the fastest computers. Since it is widely believed, though not proved, that no faster methods of solution can ever be found for these problems, NP-complete problems are considered the easiest hard problems.[5] However, some NP-complete problems have highly efficient near-optimal solution techniques, and some have good *average*-time behavior, that is, the instances that occur most often can be efficiently solved. Exponential time-hard problems, on the other hand, do not succumb to these clever methods. As an anonymous reviewer noted, this apparent gap between theoretical and practical intractability does not invalidate complexity analysis. Rather, it makes complexity analysis all the more valuable as a necessary first step on the path to efficient solution techniques. And, as is the case here, the only way to eliminate unnecessarily powerful aspects of a formal system such as GPSG is to use complexity theory.

Complexity classifications are established with the proof technique of reduction. A *reduction* converts instances of a problem T of known complexity into instances of a problem S whose complexity we wish to determine. The reduction operates in polynomial time (and in logarithmic space if T is in

[5] For additional details, the reader may refer to Lewis and Papadimitriou (1978); Garey and Johnson (1979); or Barton, Berwick, and Ristad (1987). This last work concentrates on the relationship between computational complexity and natural language.

$\mathcal{P}$). Therefore, if we had a polynomial time algorithm for solving S, then we could also solve T in polynomial time, simply by converting instances of T into S. (This follows because the composition of two polynomial time functions is also polynomial time.) For instance, if we choose T to be NP-complete, then the polynomial time reduction from T to S shows that S is at least as hard as T, or NP-hard. If we were also to prove that S was in $\mathcal{NP}$, then S would be NP-complete.

3.2 Four Classes of Computation Trees

In this paper, the problems of known complexity are based on a class of bounded computation trees. A *computation tree* is a possibly infinite tree of OR-nodes and AND-nodes, each of which contains a Turing machine configuration. That is, each node contains a state, tape contents, and a head position. A configuration C immediately dominates its successors—those configurations reachable in one machine step from C. Each computation tree completely represents the actions of a given alternating Turing machine on a given input, as explained in appendix A.1.

We are particularly interested in the four classes of computation trees that correspond to the complexity classes $\mathcal{P}$, $\mathcal{NP}$, PSPACE, and EXPPOLY. These four classes of computation trees are defined by restrictions on space (that is, configuration size), tree depth (depth is a proxy for time), and the type of branching allowed (see figures 1–4). By providing this conceptual typology, I hope to provide the reader with an intuitive, functional understanding of complexity classification.

Computation Tree for $\mathcal{P}$. The computation tree for $\mathcal{P}$ is simply a straight line containing a polynomial number of configurations (see figure 2). To see why, consider the sequence of configurations a deterministic Turing machine moves through on its way to successfully recognizing an input string x within a polynomial time bound $p(|x|)$. The machine starts in some initial configuration C_0 (read head at some starting position, blank read/write work tape, input x written on read-only input tape, and finite-state control in an initial state). Then, because it is deterministic, it moves through configurations C_1, C_2, and so forth until it reaches a final (accepting) configuration. We may therefore picture the configuration sequence as a straight line. The machine recognizes the input string x if and only if such a deriva-

Complexity Class	Computation Tree Restrictions		
	Depth	*Space*	*Branching*
$\mathcal{P}$	polynomial	polynomial	none
$\mathcal{NP}$	polynomial	polynomial	OR
PSPACE	polynomial	polynomial	AND/OR
EXP-POLY	unbounded	polynomial	AND/OR

Figure 1: The complexity classes $\mathcal{P}$, $\mathcal{NP}$, PSPACE, EXP-POLY are characterized by computation trees with restricted depth, space, and branching.

tion sequence exists.

A polynomial time DTM computation can use at most polynomial space, and therefore the configurations themselves may require polynomial space to represent their tape contents.

Computation Tree for $\mathcal{NP}$. The computation tree for $\mathcal{NP}$ is an OR-tree of polynomial depth (see figure 3). This is so because an accepting computation sequence for a polynomial-time-bounded nondeterministic TM looks like an OR-tree of polynomial depth that is rooted at the initial configuration C_0. At any step, the machine can take one of a finite number of nondeterministic branches, leading to new next-state configurations; these configurations in turn may branch. A computation succeeds if there is *any* path from the root C_0 to a final (accepting) configuration somewhere on the fringe of the tree. It is possible that some of these paths may fail or never terminate, but for the machine to recognize an input, only one sequence needs to reach an accepting configuration after some finite number of steps. There is another way of saying the same thing. We may imagine that a final configuration labels itself **true**, while any other node propagates the value **true** upward if any daughter has it. Then the computation succeeds if the root somehow ever becomes labeled **true**. In this picture, all the tree nodes are OR-nodes because a node gets labeled **true** if *any* of its daughters is labeled **true**. Again, each configuration requires no more than polynomial space to write down.

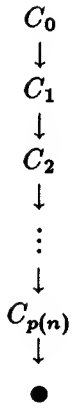


Figure 2: The computation tree for $\mathcal{P}$ is simply a straight line containing a polynomial number $p(n)+1$ of configurations.

Computation Trees for PSPACE and EXP-POLY. The computation tree for PSPACE is an AND/OR tree of polynomial depth, while the computation tree for EXP-POLY is an AND/OR tree of *unbounded* depth and polynomial size. Levels of AND-nodes and OR-nodes alternate in an AND/OR tree. (As before, an OR-node is labeled **true** if any of its daughters are labeled **true**, but an AND-node is labeled **true** only if *all* of its daughters are labeled **true**.)

These equivalences are due to a famous theorem of Chandra, Kozen and Stockmeyer (1981) which relates space and depth in AND/OR computation trees to depth and space in nonbranching computation trees.[6] Their theorem states (i) that unbounded depth AND/OR computation trees using space $s(n)$ are equivalent to nonbranching computation trees of depth $c^{s(n)}$, for some constant c, and (ii) that depth $d(n)$ AND/OR computation trees are equivalent to nonbranching computation trees using space $d(n)$. An earlier result is that OR computation trees are equivalent to nonbranching computation trees when both are allowed unbounded depth and polynomial space.

It is important to realize that restrictions on size, depth, breadth, and branching interact: the beauty of the Chandra–Kozen–Stockmeyer result

[6]The theorem was first published in 1976, in *Proceedings of the 17th Annual Symposium on Foundations of Computer Science*, on pages 89–108.

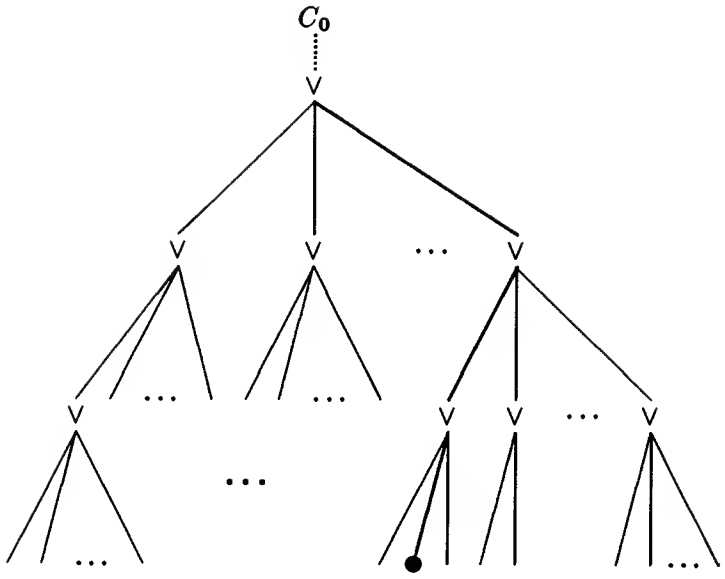


Figure 3: A computation tree for $\mathcal{NP}$ is an OR-tree of polynomial depth. The computation succeeds if *any* accepting state can be reached, as indicated by the OR-symbols (V) in the tree nodes. Here, the accepting state is symbolized by a large dot and the accepting path is marked by a dark line. There could be more than one accepting state.

is that it relates these seemingly independent restrictions. For example, a depth $d(n)$ computation tree cannot use more than space $d(n)$ because a Turing machine can access at most one tape square per move; and a depth $d(n)$ bounded AND/OR computation tree can have breadth proportional to $c^{d(n)}$ for some constant c.

In the next section, we define four problems that give insight into the computational structure of GPSG theory. To determine the complexity of these problems, we will try to find a structural equivalence between one of our four classes of computation trees and the GPSG problem S. Without loss of generality, we restrict our attention to binary branching computation trees. For our purposes, a structural equivalence is a polynomial time reduction, that is, an efficient algorithm for converting a class T of computation trees into our problem S. By finding such an equivalence between a class T of computation trees and our problem, we will have efficiently reduced T to

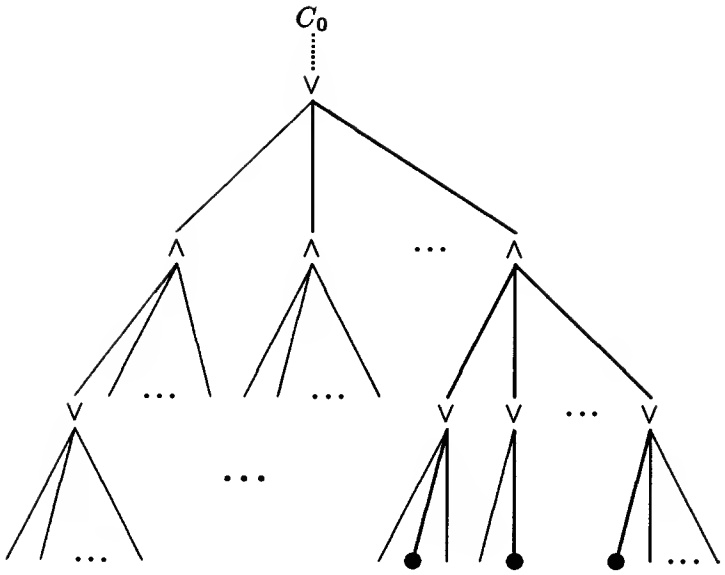


Figure 4: PSPACE and EXP-POLY computation trees are polynomial space bounded AND/OR trees. PSPACE computation trees have polynomial depth, while EXP-POLY trees have no depth bound. The subcalculation at an OR-level (∨) succeeds if any of its daughters succeed just as in figure 3, but the subcalculation at an AND-level (∧) does not succeed unless *all* of its daughters do. In this tree, accepting states are symbolized by large dots and the essential branches of the computation—making up a *pruned computation tree*—are marked by dark lines. Note that the rightmost daughter of C_0 is an AND-node, which requires every daughter to succeed.

[redacted]

4 Sources of Intractability in GPSG

This section applies our newly minted reduction technique based on computation trees to the GPSG formal system, and thereby aspires to reveal the essential intuitive character of GPSG's complexity.

We begin by examining the computational complexity of two components of the GPSG formal system (metarules and the feature system) and show how each of these systems can lead to computational intractability. Then we prove that the universal recognition problem for GPSGs is EXP-POLY hard, and hence assuredly intractable. In another words, the fastest recognition algorithm for GPSGs can take more than exponential time.

These results may appear surprising, given GPSG's weak context-free generative power. The goal of this section is to resolve this apparent paradox and answer the important computational and linguistic questions raised by the proofs: why GPSG-Recognition is so difficult, what aspects of the GPSG formalisms cause intractability, and whether they are linguistically necessary.

4.1 Introduction to GPSG Complexity

The GPSG's intractability is rooted in its formal attack on the very real problem of descriptive adequacy. In GPSG, formal devices overgenerate in order to capture the vast array of cross-linguistic phenomenon, and then constrain the overgeneration to capture exactly the phenomenon of a chosen natural language. Thus, one might almost be able to write one GPSG that simultaneously generated all natural languages. Furthermore, the components of GPSG theory can interact with each other in very powerful ways. For example, one can write ID rules that can access the same linguistic relations that UFI does, and thereby affect the operation of UFI.

The final issue I touch on before launching into a complexity analysis of the GPSG formal system is how we might best choose computational problems with which to study the GPSG formal system.

The power of a linguistic theory must be known precisely in order to meet the competing demands of descriptive adequacy and explanatory power, and to fully understand the theory. This fundamental question in mathematical linguistics is answered by measuring the power of the grammars licensed by a

linguistic theory. The power of a grammar is the difficulty of characterizing its output. The corresponding formal problem is the *recognition problem* (RP): Is a given string in a given language or not? Alternately, defining the output of a grammar to be a set of structural descriptions results in the *parsing problem*: what structural descriptions are assigned by a given grammar to a given string?

A language may be characterized in extension, by all the grammars that generate it, or constructively, by a particular grammar that generates it. In the first case, the *fixed language RP* (FLRP) is posed: Given an input string x, is x in L for some fixed language L? It does not matter what grammar generates L: both grammar and langauge are fixed (that is, ignored) in the problem statement. In the second case, the grammar is of interest, and the *universal RP* (URP) is posed: Given a grammar G and an input string x, is x in $L(G)$? Because the URP determines membership with respect to a particular grammar, it more closely models the parsing problem, which must use a grammar to assign structural descriptions.

A central goal of this work is to expose the structure of the computations specified by GPSG models. In scientific analysis, we strive to make the assumptions and generalizations that give the best insights, and hence chose our computational problems by the same criterion. *The FLRP does not lead to any insights*, and therefore we choose to study the gross power of the GPSG formal system using the URP. Barton, Berwick, and Ristad (1987), hereafter BBR, defends the *a priori* desirability of this choice.

In order to obtain still sharper insights, we must pose computational problems that capture the detailed internal organization of the GPSG formal system; for these insights to be relevant, our problems must be problems that GPSG was "designed" to solve. Recall that the process of assigning structural descriptions to utterances consists of two conceptual steps in GPSG: the *projection* of ID rules to local trees and the *derivation* of utterances from nonterminals, using the local trees. For these reasons, our complexity analysis begins by analyzing the complexity of two subproblems of GPSG projection (category projection and metarule finite closure) and one subproblem of derivation (unordered local tree recognition), and ends by analyzing the URP for GPSGs.

The complexity results are established by first exhibiting a structural equivalence between a computation tree and a GPSG formal object (for example, a category or parse tree) and then showing how the GPSG object

can be specified in polynomial time by the reduction. For each reduction, I attempt to explain the complexity results in terms of excesses in the GPSG formalisms, and motivate computational restrictions on those formal devices. The success of the next section's attempt to improve GPSG's computational and linguistic properties depends crucially on this section's success in tracing the complexity results directly back to fine details of the structure of the formal system.

Three caveats are in order. First, it is difficult to translate a precise mathematical proof into a more easily understood conceptual argument without introducing ambiguity and apparent inconsistency. Although I have strived to minimize such contagion, it surely exists, and the alert disgruntled reader is urged to seek out the formal proofs, some of which may be found in the appendices to this essay, and others in chapters 6–8 of BBR.

Second, in the GKPS grammar for English, syntactic categories are based on fixed sets and a fixed function ρ given in an appendix to their work. As such, the set K of permissible categories specified in GKPS is finite, and a large table containing K could, in principle, be given. This suggestion is of no practical significance because the GKPS category system contains at least 10^{775} categories (Ristad 1986:31). In order to better understand the computational structure of GPSGs, we must prevent such an uninformative solution. To do this, we must generalize the system of GKPS to an infinite class of GPSGs where each GPSG in the class is of finite size, containing a fixed set of ID rules, a fixed set K of syntactic categories, and so forth. Every GPSG constructed by a reduction is finite with fixed set K—none have an infinite or unbounded number of features or feature values. These generalizations are uncontroversial in theoretical computer science.

Third, for each of the four problems discussed below, many other formal restrictions would suffice to eliminate the intractability they pinpoint. For example, all parts of the GPSG projection operation can be arbitrarily bounded by (large) constants, in which case the projection operation could in principle be performed in constant time, although the constant could be large enough to prevent any physically realizable computer from ever calculating the projection operation. I focus on linguistically plausible restrictions with practical consequences, and defend the anointed restrictions in the next section.

4.2 Category Projection

To understand how FCRs and FSDs affect the cost of projecting ID rules to local trees, we define the category projection problem as follows. Let *Feat* be a set of feature-names, **Atom** a set of atomic-valued feature-names, A a set of atomic feature-values, and ρ a function from **Atom** to A. In the *category projection problem*, we are given a category C, a set R of FCRs, and a 4-tuple $\langle Feat, \mathbf{Atom}, A, \rho \rangle$ that specifies a set K of syntactic categories. The problem is to decide if the category C or any legal extension of C is in the set K. This problem will help us understand the effects of FCRs on the GPSG projection operation.

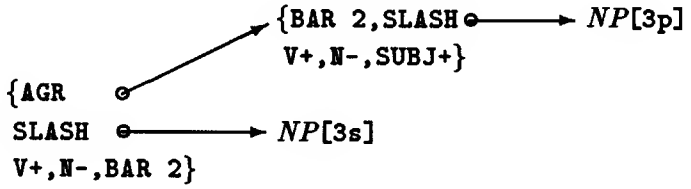
Alternately, if we were not interested in ID rule projection or about how FCRs interact with the rest of the GPSG formal system, we might study the trivial *category verification problem*, which is to decide if a given category C satisfies a given set of FCRS, as Gazdar et.al. (1988) have done.

Theorem 1 *Category Projection is PSPACE-hard.*

Proof. GPSG categories can easily be understood as trees. The atomic-valued features in a category represent a node in the tree, and a category C dominates its embedded categories—that is, C immediately dominates all categories C' such that for some category-valued feature f, $C(f) = C'$. For example, the GPSG category

$$V2[\mathbf{AGR}S/NP[3p]]/NP[3s]$$

corresponds to the tree



In our reduction, then, we can use categories to represent a binary branching AND/OR computation tree. 0-level categories represent the nodes

of the computation tree: atomic-valued features $f_1, f_2, \ldots, f_n$ encode the machine state, tape contents, head position, branching type, and truth value. Category-valued features encode domination in the computation tree: the two category-valued features LEFTi and RIGHTi represent the left and right branches of the computation tree at level i. For example, in the entire category C_1 constructed by our reduction, the top-level 0-category encodes the root node of the computation tree, and the the values C_1(LEFT1) and C_1(RIGHT1) encode the left and right branches, respectively, of the root node (initial configuration). Although the category C_1 is undefined for any other category-valued features, the immediately embedded categories C_1(LEFT1) and C_1(RIGHT1) are each defined for exactly two category-valued features: LEFT2 and RIGHT2. Continuing in this manner, we will need exactly two category-valued features for each level, to represent the two of subtrees immediately dominated by each node at a given level:

$$
\begin{array}{l}
\{f_1, f_2, \ldots, f_n, \\
\quad [\texttt{LEFT1}\ \{f_1, f_2, \ldots, f_n, \\
\qquad\qquad [\texttt{LEFT2}\ \{\ldots\}], \\
\qquad\qquad [\texttt{RIGHT2}\ \{\ldots\}]\}], \\
\quad [\texttt{RIGHT1}\ \{f_1, f_2, \ldots, f_n, \\
\qquad\qquad [\texttt{LEFT2}\ \{\ldots\}], \\
\qquad\qquad [\texttt{RIGHT2}\ \{\ldots\}]\}]\}
\end{array}
$$

FCRs maintain the next-move relation between an embedded category C_i at level i and the two categories it immediately contains C_i(LEFTi) and C_i(RIGHTi) and calculate the truth labelings of internal nodes according to branching type.

In order for such a reduction to be polynomial time, we must be able to write down a a set of FCRs and to specify the set K of syntactic categories in polynomial time. To specify K, we write down a function ρ and the sets *Feat* of features, Atom of atomic-valued features and A of atomic feature values. The restriction to polynomial time reductions means we can only construct a polynomial number of features: because atomic features represent configuration size, we can only represent polynomially-sized configurations; because category-valued features represent depth and the finite feature closure restriction prevents a category-valued feature from dominating itself, we can only represent polynomially deep computation trees. Therefore, we can use category projection to simulate any polynomial size and depth AND/OR computation tree: the category projection problem is PSPACE-hard. □

Let us now restrict our attention to 0-level categories.

Theorem 2 *0-Level Category Projection is NP-hard.*

Proof. In order to prove this theorem 2, we must hide an entire depth $d(n)$ OR computation tree in a 0-level category. Rather than use the entire set **Atom** to encode one node of the computation tree (a Turing machine configuration) as above, we will use the set **Atom** to encode the entire pruned OR computation tree. To do this, we partition the set **Atom** into $d(n)$ subsets $\mathbf{Atom}_1, \mathbf{Atom}_2, \ldots, \mathbf{Atom}_{d(n)}$, each of size $d(n)$. (Recall that the nodes of a computation tree can be no larger than the depth of that tree.) The atomic features in $\mathbf{Atom}_i$ will encode the i^{th} node of the computation tree. Each fully specified 0-level category represents a polynomial number of polynomial size Turing machine configurations, instead of merely one such configuration as in the preceeding proof. This can be done in polynomial time because a polynomial times a polynomial is also a polynomial. Finally, a polynomial number of disjunctive consequence FCRs relate the features $\mathbf{Atom}_i$ representing node i to the features $\mathbf{Atom}_{i+1}$ representing the successor node $i+1$ according to the next-move relation of the OR computation tree.

In such a manner it is possible to hide an entire pruned OR computation tree in a 0-level category, and prove that the 0-level category projection problem is NP-hard. □

Note, however, that the proof of this theorem is much clearer when the NP-complete Satisfiability problem is used instead of a node-bounded computation tree. Ristad (1986) contains such a proof.

Restricting the Theory of Syntactic Features. As we just saw, the primary computational resource provided by the theory of syntactic features is polynomial space. This arises from finite feature closure, which generates a surprisingly large number of possible syntactic categories. Ristad (1986) observes that even if all atomic-valued features are restricted to be binary-valued, finite feature closure admits $\theta(3^{a \cdot b!})$ GPSG categories where a is the number of atomic-valued features and b the number of category-valued features. In fact, there are more that 10^{775} categories in the GKPS system.

Fortunately, the full power of embedded categories does not appear to be linguistically necessary because no category-valued feature need ever contain another in an ID rule. To be precise, although a category-valued feature f

may appear inside another category-valued feature g in the parse tree of some utterance in some language, f will never be *required* to appear inside g in a rule of any natural language grammar. In GPSG, there are four category-valued features: SLASH, which marks the path between a gap and its filler with the category of the filler; AGR, which marks the path between an argument and the functor that syntactically agrees with it (between the subject and matrix verb, for example); WH, which marks the path between a *wh*-word and the minimal clause that contains it with the morphological type of the *wh*-word; and RE, which marks the path between an anaphor and its antecedent with the category of the anaphor.

No category-valued feature f need ever contain a category-valued feature g because (i) for foot features, the path that g marks need never be extended by the path that f marks: g could just as well cover a longer path containing both it's former path and the path of f, and (ii) for the head feature AGR, the path that f marks need never be dependent on the path a foot feature g marks. Specifically, RE will not contain a category-valued feature, because the value of RE is the category of an anaphor and anaphors are nominals without internal phrase structure. AGR will never contain SLASH or RE because a functor (verb or predicate) will never select a gap, a constituent containing a gap, or a category that must be an antecedent to some unknown anaphor as its argument. SLASH need never contain RE because the path between a gap and it's filler is never dependent on the path between an anaphor and its antecedent; SLASH will never be required to contain AGR because such a category corresponds to "the following imaginary (and rather weird) case: Suppose we found a language in which finite verb phrases could be fronted over an unbounded domain provided that they were in the agreement form associated with third-person-singular NP controllers" (Pullum, personal communication). Finally, because the value of WH is the category of a *wh-* noun phrase, and because *wh-* nominals are never required to contain gaps, WH need never contain SLASH or AGR. In point of fact, no category embeddings appear in the GKPS grammar for English, and it is difficult to see why they would be necessary for any other natural language.[7]

[7]The central *apparent* counterexample to these arguments is cases of multiple extractions (see footnote 3 above). Both GPSG and RGPSG must be modified in order to handle this phenomenon. One way is to abandon closure constraints on category embeddings, in order to allow SLASH to take as its value a category specified for SLASH (so-called "recursive SLASH"). Such a change would, in my opinion, be disastrous because it abrogates feature closure, a central descriptive and computational constraint in the GPSG category system. The revised principles of UFI needed to constrain recursive SLASH would, I suspect, be

Let us now explicitly adopt the strategy of restricting computational costly devices in the absence of direct linguistic counterevidence. This strategy will result in the most constrained and falsifiable theory. The obvious revision, then, is *unit feature closure*: to limit category-valued features to containing only 0-level categories. This revision makes categories and 0-categories equivalent for the polynomial time reductions of computational complexity theory.

Restricting Marking Conventions. Satisfying FCRs and FSDs in the GPSG category projection process requires significant computational resources. First, each device allows the projection process to reuse the polynomial space provided by the theory of syntactic features, because each can establish equivalences between the features in a category C and the features in a category embedded in C. This ability to apply across embedded categories vastly increases the complexity of the rule-to-tree projection. To see why it is linguistically unnecessary, consider the role of embedded categories. A category-valued feature f expresses a nonlocal linguistic relation between a category C and the one or more connected categories that bear the feature specification $[f\ C]$. Thus, in the linguistically relevant cases, every embedded category eventually "surfaces" in phrase structure, where the marking conventions are free to apply. The one exception to this argument is FCR 13 in the GKPS grammar for English, which applies 'across' an embedded category.

FCR 13: [FIN, AGR *NP*] ⊃ [AGR *NP*[NOM]]

In RGPSG, marking conventions may not apply to or across embedded categories. The effect of FCR 13 is achieved in RGPSG by a combination of carefully written ID rules and the simple default SD 2 in section 6.2 below.

very tricky to state. A simpler approach is to allow SLASH to take a strictly bounded sequence of values, where a category preceeds all categories in the sequence if and only if its extraction path properly contains their extraction paths. Then the revised FFP might enforce the path containment condition by limiting ID rules to only affecting (adding or removing) the last category in the SLASH sequence on a local tree. (That is, feature instantiation in ID rule projection can only add categories to the front of the SLASH sequence.) A still simpler, although seemingly less elegant, approach is to add a new foot feature, say SUBSLASH, to encode an extraction path contained inside the extraction path marked by SLASH.

Second, FCRs and FSDs of the "disjunctive consequence" form $[f\ v] \supset [f_1\ v_1] \vee \cdots \vee [f_n\ v_n]$ allow us to simulate the next-move relation of an OR computation tree; from another perspective they compute the direct analog of the NP-complete Satisfiability problem: when several such FCRs are used together, the GPSG must nondeterministically try all n feature-value combinations.

Third, the process of applying feature specification defaults to local trees is very complex, in part because it is not informationally encapsulated. Rather than simply considering the (existing) feature specifications in each target category separately, FSD application is affected by the other categories in the ID rule, all principles of universal feature instantiation, and even FCRs.

There is no reason to believe that marking conventions need be as powerful and unconstrained as FCRs and FSDs. The approach RGPSG takes is to virtually eliminate marking conventions. Rather than stating the internal constraints on categories explicitly (and redundantly), as FCRs do, RGPSG eliminates FCRs altogether. Instead, the constraints FCRs express are implicitly stated in the rest of the grammar — in the way ID rules and metarules are written, for example.

Reducing the power of marking conventions and other grammatical devices might appear to hinder the grammar writer. But we help the grammar writer understand the consequences of a grammar by reducing the complexity of grammatical derivations: if a supercomputer cannot possibly determine membership in the language of an intractable grammar, then surely no human grammar writer can—see appendix A.3 for empirical confirmation. Evans (1985:213) observes exactly this practical consequence of GPSG's theoretical intractability: "The GPSG theory is complex enough that ensuring that any but a small grammar behaves as expected is a difficult task."

4.3 Metarule Finite Closure

In the *finite closure membership* problem for GPSG metarules, we are given an ID rule r, a set M of metarules, and a set R of ID rules. The problem is to decide whether r is in $FC(M, R)$. This subproblem of GPSG projection will allow us to pinpoint the contribution of metarules to the complexity of ID rule projection.

Theorem 3 *Metarule Finite Closure Membership is NP-hard.*

Proof. The insight underlying the following reduction is that the finite closure operation specifies an OR computation tree whose nodes are ID rules, where metarules enforce the next-move relation between adjacent nodes. The polynomial-time reduction restriction limits us to constructing at most a polynomial number of metarules, while metarule finite closure restricts each metarule to at most one application in a given computation tree: metarules by themselves are only capable of simulating a polynomial depth computation tree. The finite closure operation on metarules gives the GPSG formal system the power of nondeterminism (OR branching) because all possible permutations of metarules are applied; using this power, we can use a metarule system to simulate any polynomial depth OR computation tree. Therefore, the metarule finite closure membership problem is NP-hard. □

Restricting Metarules. Metarule finite closure generates many linguistically incorrect ID rules that must be excluded by other GPSG devices, such as FCRs. For example, the result of applying the Extraposition, Passive, and Subject-Aux Inversion metarules in order to the lexical ID rule 2

$$VP\,[\texttt{AGR}\ S] \rightarrow H\,[20], NP \tag{2}$$

is the lexical ID rule 3,

$$S\,[\texttt{+INV, PAS, AGR}\ NP[\texttt{it}]] \rightarrow H[20], S, NP, (PP\,[\texttt{by}]) \tag{3}$$

which does not generate sentences in the English language and (thankfully) is excluded by FCR 1.

The GKPS grammar for English contains six metarules; out of approximately 1944 possible metarule interactions in principle, only two such interactions appear to be productive (passive followed by subject-aux inversion or slash termination metarule 1).[8] Therefore, the second metarule restriction adopted by RGPSG is *biclosure*, instead of finite closure. Alternately,

[8] Given a set of n metarules, the number of possible metarule interactions is the number of ways to pick n or less metarules from the set, where order matters and repetitions are not allowed. That number is given by the total number of possible k-selections from the n metarules, where k varies from 0 (no metarules apply) to n (any combination of all metarules apply). Thus, the number of possible interactions $f(n)$ is: $\sum_{k=0}^{n} \frac{b!}{(b-i)!} \approx b! \cdot e$). Note that this is *not* the size of metarule finite closure, because it does not consider the possibility of a metarule matching an ID rule in more than one way.

we might restrict metarules to unit closure, or follow Pollard (1985) in eliminating metarules altogether.

Lacking an alternate theory of lexical redundancy rules, honesty compels us to include metarules in RGPSG in some form. How then can we choose between unit closure and biclosure on principled grounds? Metarule biclosure does not overgenerate as badly as finite closure, and thereby promotes descriptive adequacy at the expense of some explanatory power. Biclosure has an edge in descriptive economy over unit closure because simpler (and fewer) metarules are needed with biclosure than with unit closure. Thus, although the length of metarule derivations is not subject to direct empirical evidence, it is not entirely *ad hoc* because it is subject to the scientific criterion of descriptive economy, descriptive adequacy, and explanatory power.

4.4 Unordered Local Tree Recognition

The universal recognition problem for unordered local trees is to decide if a given string x can be derived from a set of unordered local trees P. This is equivalent to the unordered context-free grammar recognition problem considered by E. Barton in BBR.

Theorem 4 *Unordered Local Tree Recognition is NP-complete.*

This is the only theorem we will not prove here using our typology of computation trees. Barton shows how the multiset RHS of an ID rule contributes to an exponentially large space of local phrase structure trees: an ID rule with a a RHS of cardinality b can, if unconstrained by LP statements, correspond to $b!$ ordered productions. In parsing practice, this can cause a combinatorial explosion in a context-free parser's state space. In addition to causing nondeterminism in any GPSG-based parser, the multiset RHS confers on GPSG the ability to count nonterminals. The apparent artificiality of this device, as discussed in BBR (pp.260-261), will motivate RGPSG to adopt a substantive constraint of *short ID rules* (binary branching, for example).[9]

[9]The binary branching constraint is independently motivated by the linguistic arguments of Kayne (1981) and others. In that work, Kayne argues that the path from a governed category to its governor (for example, from an anaphor to its antecedent) must be unambiguous—informally put, "an unambiguous path is a path such that, in tracing

4.5 GPSG Recognition

The ultimate problem we analyze is the universal recognition problem for GPSGs: given a GPSG and a string, is the string in the language of the GPSG? (Recall that the URP was chosen as the problem statement that best characterized the overall power of a grammar.)

Theorem 5 *GPSG Recognition is EXP-POLY-hard.*

Proof. The idea of this reduction is to transparently encode a pruned polynomial space AND/OR computation tree in a GPSG parse tree. Every node in the computation tree will be represented by a category in the GPSG parse tree. The reduction preserves the invariant that a GPSG category can be terminated if and only if the configuration that it represents can be labeled **true** in the pruned computation tree.

As before, 0-level categories encode nodes of the computation tree, which are Turing machine configurations. Local trees represent the *pruned* next-move relation between nodes: a local tree with one daughter represents a pruned OR node, while a local tree with two daughters represents a pruned binary-branching AND node. The leaves of the pruned computation tree have halted, accepting configurations; these accepting nodes are represented by a local tree with no daughters. There are no lexical entries in this GPSG, and therefore the only way to terminate a category in this GPSG is with such a "null transition." Thus, the GPSG parse tree will terminate in a very long empty string.

Now we must show how such an exponentially large parse tree can be specified in polynomial time. The reduction must first list enough atomic features to represent the largest node in the computation tree; this is possible because the size of each node is bounded by a polynomial, as is the reduction. We will not be able to write down all the local trees required in polynomial time, because there are an exponential number of them. (In fact, approximately c^3 local trees are needed, where c is the number of possible

---

it out, one is never forced to make a choice between two (or more) unused branches, both pointing in the same direction" (Kayne 1981:146). The unambiguous path requirement sharply constrains fan-out in phrase structure trees because n-ary branching, for $n > 2$, is only possible when none of the n sister nodes must govern any other nodes in the phrase structure tree.

configurations, which we know to be exponential in the polynomial size of the configurations.)

Instead, we will use ID rules to encode the alternating Turing machine transition relation δ, which is infinitely smaller than the corresponding next-move relation. Recall that δ is a relation between a tuple and a triple: the tuple contains a machine state and currently scanned tape symbol, while the triple contains a new machine state, a new symbol to write on the tape, and a direction to move the read/write head. The next-move relation is a relation between two configurations that obey the δ relation. Each transition in δ licenses infinitely many next-move relations between nodes of the computation tree because δ does not care about tape squares that the machine is not currently scanning. For every binary OR transition licensed by δ, we will build two nonbranching ID rules $C \rightarrow C_1$ and $C \rightarrow C_2$, one for each of the two possible pruned OR transitions (recall that a pruned OR branch is a straight line). For every binary AND transition found in δ, we will build a branching ID rule $C \rightarrow C_1, C_2$. Therefore, an OR category can be terminated iff one of it's possible daughters can be, while an AND category can be terminated iff all of its daughters can be. This corresponds exactly to the labeling rules for an AND/OR computation tree: an OR node is labled true iff one of its possible daughters is, while an AND node is labelled true iff all of its daughters are.

Next, we add an lone ID rule $C_{accept} \rightarrow \epsilon$ to terminate nodes representing halted, accepting configurations with the empty string. Because there are no lexical entries in this GPSG, the only categories that can be terminated are those that represent nodes that have been labeled true in the computation tree.

The final step is to make the atomic features that represent the tape contents be head features, and insist all daughters be heads. An ID rule that encodes a δ-transition will then project into the local trees that represent all possible next-moves licensed by that δ-transition. The head feature convention, which governs the projection of ID rules into local trees, will ensure that tape squares not altered by the tape-writing activity specified by the δ-transition will be identical on the mother and all daughters in the projected local trees. In this fashion we can use a GPSG to simulate any unbounded depth polynomial space AND/OR computation tree. Therefore, GPSG Recognition is EXP-POLY time-hard (details are in appendix A.1). □

What are the implications of this result for GPSG and natural language? At first glance, it is unclear whether we have exposed an oversight in the way GPSG was formalized (and if so, how easily may it be remedied?) or an inherent property of natural language grammars. Equally unclear is whether the intractability arises in practice or is merely an artifact of the complexity-theoretic idealization to unbounded inputs. But first we reconcile this result with the fact that context-free languages may be recognized in polynomial time.

4.5.1 Interpreting the Result

At first glance, a proof that GPSG Recognition is EXP-POLY hard appears to contradict the fact that context-free languages can be recognized in $O(n^3)$ time by a wide range of algorithms. To see why there is no contradiction, we must first explicitly state the argument from weak context-free generative power, which we will call the *efficient processability* (EP) argument.

The Efficient Processability Argument. The main thrust of the EP argument runs as follows:

- Any GPSG can be converted into a weakly equivalent context-free grammar (CFG).
- CFG recognition can be accomplished in polynomial time.
- Therefore, GPSG recognition can also be accomplished in polynomial time.

The argument continues:

- If the conversion is fast, then GPSG recognition is fast. However, even if the conversion is slow, recognition with the "compiled" CFG will still be fast; we may justifiably lose interest in doing recognition with the original, slow GPSG.

The EP argument is misleading because it ignores both the effect conversion has on grammar size and the effect grammar size has on recognition speed. Crucially, grammar size affects recognition time in all known CFG

recognition algorithms. The only grammars *directly* usable by context-free parsers—hence the only grammars for which rapid parsing results carry over—are those composed of *context-free productions* with *atomic nonterminal symbols*. For GPSG, this corresponds to the set of admissible local trees, and this set is astronomical. Ignoring the effects of ID/LP format, it is

$$\theta(3^{m!m^{2m+1}}) \tag{4}$$

in a GPSG G containing m symbols (see BBR for details).

This worst-case formula for the size of the "expanded" grammar is vindicated in practice. Phillips and Thompson (1985:252) observe that in their parser based on the GPSGs of Gazdar (1982), "To expand the [GPSG] grammar completely ... would be ridiculously wasteful of space and time. (The toy grammar of English we use with GPSGP [their parser], of 29 phrase-structure rules and four metarules, which expands to 85 rules, is equivalent to several tens of millions of context-free rules.)" Similarly, Shieber (1983:137) notes that typical post-Gazdar (1982) GPSG systems contain "literally trillions" of derived rules. In appendix A.2, I estimate that the GKPS grammar for English projects to more than 10^{33} admissible local trees.

Consequences for GPSG Parsing. The Earley recognizer for context-free grammars runs in time $\theta(|G'|^2 \cdot n^3)$ where $|G'|$ is the size of the CFG G' and n the input string length, so a GPSG G of size m will be recognized in time

$$\theta(3^{2 \cdot m!m^{2m+1}} \cdot n^3) \tag{5}$$

The hyperexponential term will dominate the Earley algorithm complexity in the reduction above because m is a function of the size of the ATM we are simulating. Even if the GPSG is held constant, the stunning derived grammar size in formula (4) turns up as an equally stunning "constant" multiplicative factor in (5), which in turn will dominate the real-world performance of the Earley algorithm for all expected inputs (that is, any that can be written down in the universe), every time we use the derived grammar. This class of hyperexponential functions c^{n^n} grows at a frightening rate—in the mathematical worst case, if a GPSG with 2 symbols recognized a given sentence in .001 second, a grammar with 3 symbols would recognize the same sentence in 2.5 hours, and a grammar with a mere 4 symbols could take at least 10^{63} *centuries*.

GPSG's intractability appears in GPSG-based parsers in two ways, strongly suggesting that the GPSG's intractability is not an artifact of complexity theory. First, many GPSG-based parsers appear to be infamously slow. For example, Evans (1985:237) experiences the real-world intractability of GPSG-Recognition first hand in his GPSG-based parser, and proposes to manage it by eliminating lexical ambiguity and by keeping both grammar and input string size as small as possible: "The attempts to overcome the time and space problems have only been partially successful .... The only remedies seem to be, keep phrases as short as possible (for example, do not try to test large noun phrases inside complex sentences if it can be avoided—use proper nouns instead), make sure no words are duplicated in the lexicon, keep the number of ID rules currently loaded down where possible ...."

Second, I know of no faithful implementation of GKPS. As we just saw, there are too many possible local trees for any computer to explicitly calculate GPSG projection, which means that parsers must project ID rules on the fly. But we know from theorems 1–4 that the GKPS projection operation cannot even be computed in practice, due to the complexity of metarules, marking conventions, embedded categories, ID rules, and exceptional feature specifications. Thus it will not be possible to faithfully project ID rules on the fly, in part because not all extensions of ID rule categories are legal. For example, the categories *VP*[INV +,VFORM PAS] and *VP*[INV +,VFORM FIN] are not legal extensions of *VP* in English due to FCR 1, while *VP*[INV +,AUX +,VFORM FIN] is. But even if we ignore the significant computational complexity introduced by syntactic features, marking conventions, ID/LP format, null-transitions, and metarules, *the GPSG recognition and projection problems will still be intractable.* This is because the head feature convention alone allows GPSG parse trees to simulate unbounded depth polynomial space nonbranching computation trees: the recognition problem for these impoverished GPSGs would be PSPACE-hard and still thought to be intractable. (This result should not surprising, because the HFC in current GPSG theory replaces some metarules in earlier versions of GPSG and metarules are known to cause intractability.) Because no faithful implementation of GKPS is even possible in practice, computational linguists have no choice but to in effect invent their own version of GPSG theory to implement.[10]

[10] One such parser for English derived from GPSG is described in Harrison and Maxwell (1986), who claim that "parser response time has been adequate for our development purposes."(p.10) However, they note that "in the presence of significant ambiguity, an

I am not impugning any of these GPSG-based parsing systems. Rather, I am arguing that GPSG's theoretical intractability is not an artifact of complexity theory because it appears in the real world, in natural language parsers based on GPSG theory, which will be as slow as they are faithful to GKPS.

As we shall see below, GPSG's intractability appears to be due partly to oversights in it's formalization and partly to an inherent property of natural language grammars. The fact that revised GPSG has the empirical coverage of GPSG, and is only NP-complete, argues that GPSG's EXP-POLY-hardness arises from the particular formal choices GKPS made. But the fact that intractability in both GPSG and RGPSG arises from the need to account for the very real linguistic phenomenon of nonlocal syntactic agreement and ambiguity suggests that all natural language grammars may be intractable (more below).

4.5.2 Restricting the GPSG formal system

The proof of theorem 5 tells us that we must further restrict the GPSG formal system, in both projection and derivation, in order to achieve computational tractability. Let us now consider how that might be done without curtailing GPSG's descriptive economy too much.

Restricting ID/LP. ID rules significantly increase the time resources required by the GPSG derivation process in three related ways. First, a derivation step is nondeterministic because a category may immediately dominate more than one RHS. Second, the derivation process may *alternate* between a derivation step involving the ID rules $C \rightarrow C_1 \mid \ldots \mid C_k$ that corresponds to an OR-transition (only one of k possible successors must yield a terminal string) and a derivation step involving an ID rule $C \rightarrow C_1, C_2, \ldots, C_k$ that corresponds to an AND-transition (all k successors must yield terminal strings). These two devices introduce lexical and structural ambiguity. As is

---

all-paths parser such as this one can experience a significant degradation [in] response time."(p.10) Their parser projects local trees on the fly, and fails to implement FCRs, FSDs, metarules, ID/LP format, and the CAP in any form. They disallow exceptional feature specifications; feature instantiation is ad hoc, and not faithful to the HFC and FFP of GKPS. Harrison (personal communication) attributes the parser's adequate response times to clever programming and it's departure from the specifics and generality of the GKPS formal system in order to avoid the formal excesses of GKPS.

well-known, ambiguity is a central property of natural languages. Therefore, this aspect of ID rules is linguistically essential, and it will be retained in RGPSG.

Third, unrestricted null transitions in ID rules are a source of intractability because they allow GPSGs to generate enormous phrase structure trees whose yield is the empty string. If there were no null transitions in ID rules, then the GPSG formal system could only simulate polynomial breadth computation trees. This is so because the polynomial time reduction can only write down polynomial length input strings x, and in a grammar free of null transitions, the length of a string is equal to the breadth of its parse tree. Unrestricted null transitions are also undesirable according to classic linguistic arguments, because GPSG theory with unrestricted null transitions need not obey recoverability of deletions (ROD). A parser that used such a grammar must nondeterministically postulate elaborate phrase structure in between its input tokens.

Although unrestricted null-transitions violate ROD and cause unnatural computational difficulties, they are absolutely needed for gaps: the RGPSG solution is to greatly restrict null-transitions by strengthening the X'-theory embodied in ID rules.

Restricting Universal Feature Instantiation. The three principles of UFI all cause intractability because they allow the derivation process to in effect reuse space resources.

First, each principle of UFI can enforce nonlocal feature agreement in phrase structure. Appendix B.1 shows how this causes NP-hardness, when coupled with lexical ambiguity or null transitions. A related source of intractability is that the projection of ID rules to local trees can create an astronomical space of local trees, which in turn increases parser search space. These two sources of intractability cannot be eliminated because they are essential to GPSG's account of linguistic agreement among conjuncts and between predicates and their arguments, gaps and their fillers, anaphors and their antecedents, and phrases and their lexical heads.

The use of exceptional feature specifications in these principles allows a derivation to reuse the space resources provided by the ID rules and theory of syntactic features. In the EXP-POLY reduction above, head features encode nodes of the computation tree. The HFC is used to transfer the tape contents

of a configuration C_0 (represented by the mother) to its immediate successors $C_1, C_2, \ldots, C_k$ (the head daughters). The configurations $C_0, C_1, \ldots, C_k$ have identical tapes, with the critical exception of one tape square. If the HFC enforced absolute agreement between the head features of the mother and head daughters, the polynomial space AND/OR computation tree could not be simulated in this manner.

Restricting Metarules. Although no metarules are involved in the EXP-POLY reduction, they can indirectly increase the time and space resources needed by the derivation process by introducing null transitions and ambiguity in ID rules.

As we noted, unrestricted null transitions are both linguistically and computationally undesirable. Moreover, the ability of metarules to affect lexical head daughters is in direct conflict with their linguistic purpose: "to express generalizations about the subcategorization possibilities of lexical heads." (GKPS:59) Unrestricted metarules can destroy the relation between a phrase and its lexical head, and thereby violate X'-theory. The first step in revising metarules is to restrict them to *only* affect the mother and nonhead daughters in lexical ID rules. Because of this change, metarules cannot alter the [NULL -] specification that appears on all head daughters in RGPSG ID rules. Therefore, once a category is expanded in an RGPSG derivation, it *must* be lexically realized in the derived string. This formal constraint ensures that the empty string does not have elaborate phrase structure in RGPSG.

4.6 Sources of Intractability Summary

Figure 5 summarizes the sources of intractability we have uncovered by applying complexity analysis to four carefully selected computational problems posed within the GPSG formal system. In the next section, I present revised GPSG. Of the more than ten sources of intractability lurking in GPSG, only two remain in RGPSG — lexical ambiguity and nonlocal feature agreement. Critically, these two sources of intractability in RGPSG appear to be linguistically essential (see Ristad and Berwick, 1989).

Syntactic Categories	◁ Finite feature closure
ID Rules	◁ Unrestricted nulls (gaps) ◁ Alternating derivations (lexical & structural ambiguity) ◁ Unbounded multiset RHS
Metarules	◁ Introduce nulls & alternation ◁ Finite closure
UFI	◁ Nonlocal agreement ◁ Exceptional feature specification
Marking Conventions (FCRs, FSDs)	◁ Disjunctive consequence ◁ Apply across embedded categories

Figure 5: Sources of Intractability in GPSG

5 The RGPSG Formal System

The revision of the GPSG formal system is driven by the desire to strengthen linguistic principles embodied in it and reduce the computational resources it uses in both projection and derivation. The common theme of the proposed restrictions is to reduce the range and interaction of RGPSG's formal devices. Specifically, RGPSG obeys stricter notions of X'-theory, recoverability of deletions, and permissible extraction domains than standard GPSG does. The computational restrictions on RGPSG focus on bounding the size, depth, breadth, and branching type of the computation trees that can be found in RGPSG's formal devices.

Recall that our strategy is to restrict the computational power of a formal device in the absence of linguistic counterevidence. As we shall see, our goal of an efficient, maximally restricted, descriptively adequate formal system is frequently at odds with the goal of a simple and notationally elegant formal system. Systems with the simplest rule format are often the least restrictive—rewriting rules, for example, are the most intractable (undecidable) when they are notationally simplest (unrestricted). We are interested in natural restrictions that eliminate unnatural grammars and result in the most efficient formal system, *not* ones that result in a simpler rule format. As Chomsky (1965:61–2) observed, "the critical factor in the development of a fully adequate theory is the limitation of the class of possible grammars. . . . we should like to accept the least 'powerful' theory that is empirically adequate."

5.1 Overview

The RGPSG process of assigning structural descriptions to utterances differs slightly from the GKPS conception. Figure 6 shows the internal organization of RGPSG projection. First, metarules and marking conventions are applied to ID rules, resulting in an enlarged set of ID rules R'. Then the rules in R' are used to derive the utterances in the language of the RGPSG. Unlike GPSG, the RGPSG derivation operation includes UFI and LP because both devices are informationally encapsulated and functionally independent. The lack of FCRs, FSDs, and exceptional feature specifications means that ID rule extension is monotonic, unlike in GPSG: every legal ID rule has an easily-computed legal extension, unlike in GPSG.

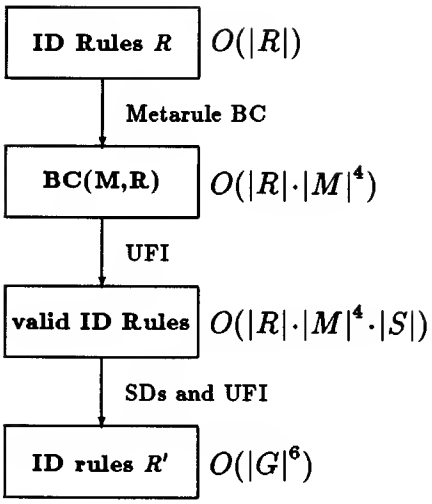


Figure 6: This diagram shows the projection of an RGPSG G with ID rules R, metarules M, and simple defaults S. The O-bounds show the effect of various formal devices on derived grammar symbol size.

5.2 Theory of Syntactic Features

The set K of RGPSG syntactic categories is specified by listing a set *Feat* of features, a set **Atom** of atomic-valued features, a set A of atomic feature values, and a function ρ that defines the range of each atomic-valued feature, as in GPSG. The major change is is *unit feature closure* instead of finite feature closure: category-valued features may only contain 0-level categories. (0-level categories do not contain any category-valued features). RGPSG adopts this strongly falsifiable constraint; the linguistic justification may be found above. The depth of category-embedding is purely an empirical issue, and hence unit closure is not *ad hoc*.

The other revision is primarily notational: any RGPSG feature f may assume the distinguished values **noBind** or **unBound** in addition to those values determined by $\rho(f)$. A **noBind** value indicates that the feature may not receive a value in any extension of the given category, while **unBound** indicates that the feature does not currently have a value, and may receive

one in extension (having a unBound value is the same as being unspecified in a unification grammar).

5.3 Immediate Dominance/Linear Precedence

According to the simplest version of X'-theory, all phrases must have heads. Although GPSG lacks a formal constraint to this effect, in point of fact every ID rule in the GKPS grammar for English has a head. For these reasons, RGPSG ID rules must have exactly one mother and at least one head daughter. The heads are separated notationally from the nonheads by a colon, and appear to the left of the colon. The mother and all head daughters are implicitly specified for [NULL -]. For example, the RGPSG headed ID rule 6 corresponds to the GPSG ID rule 7.

$$V2 \rightarrow \texttt{[SUBCAT 2]} : N2 \tag{6}$$

$$V2\,\texttt{[NULL -]} \rightarrow \texttt{H[SUBCAT 2,NULL -]}, N2 \tag{7}$$

There is only one lexical element for the null string, and it is universal across all grammars:

$$X2\,\texttt{[SLASH}\ X2_1\texttt{,NULL +]}_1 \rightarrow \epsilon \tag{8}$$

Co-subscripting indicates that the two *X2* categories must be identical in any legal projection of the rule, with the exception of the [NULL +] and SLASH specifications. This restricted ID rule format, when coupled with a restriction on metarules that prevents them from affecting head daughters, prevents head daughters from ever being erased in a RGPSG derivation. Thus, null transitions are effectively eliminated from RGPSG.

An *ordered production* is an ID rule whose daughters are completely linearly ordered, that is, a string of daughter categories rather than multisets of head and nonhead daughters. An ordered production is *LP-acceptable* if all LP statements in the RGPSG are true of it.

The RGPSG ID/LP formalism by itself does not contain formal constraints sufficient to guarantee polynomial-time recognition, although the linguistically justified use of *short ID rules* can render ID rules tractable, because ID/LP grammars with bounded rules can be parsed in time polynomial in the grammar size.[11]

[11] If the length bound for natural language grammars is the constant b, then any ID/LP

5.4 Metarules

An RGPSG metarule may *only* affect the mother and at most one nonhead daughter in a lexical ID rules. Thus, the only way a metarule can affect a head daughter is to introduce a new head feature on the mother, which will appear on the head daughter by virtue of the HFC if and only if the head daughter is unspecified for the new head feature. This is how the passive metarule operates in RGPSG:

$$\begin{array}{c} VP \rightarrow W, NP \\ \Downarrow \\ VP[\texttt{+PAS}] \rightarrow W, (PP[\texttt{by}]) \end{array} \tag{9}$$

The complete set of ID rules in a RGPSG is the maximal set that can be arrived at by taking each metarule and applying it to the set of rules that did not themselves arise from the application of that metarule or from the application of one or more other metarules. This maximal set is called the *biclosure BC(M, R)* of a set R of headed lexical ID rules under a set M of metarules.

Recall that a metarule may determine more than one ID rule per input ID rule because a metarule pattern may match an ID rule in more than one way. Given a set of ID rules R whose size is n symbols, and given a set of metarules M whose size is m symbols, the symbol-size of the unit closure $UC(M,R)$ is $\theta(n + n \cdot m^2) = \theta(|G|^3)$. Each symbol in M can, in the worst case, match each symbol in R, resulting in at most $\theta(m)$ new symbols per match. Therefore, the symbol-size of the biclosure $BC(M,R)$ is $\theta(n \cdot m^4) = \theta(|G|^5)$.

5.5 Principles of Universal Feature Instantiation

Principles of universal feature instantiation in RGPSG all preserve a simple invariant across all ID rules. They are monotonic; that is, they never delete or alter existing feature specifications. The head feature convention,

---

grammar G can be converted into a strongly-equivalent CFG G', of size $\theta(|G| \cdot b!) = \theta(|G'|)$ by simply expanding out the constant number of linear precedence possibilities. In the GKPS and RGPSG grammars for English, $b = 4$ for the result of the Subject-Aux-Inversion metarule applying to a [SUBCAT 44] rule headed by the auxiliary *be*. (I ignore the iterating coordination schema, which licenses rules with unbounded right-hand sides.)

for example, strengthens X'-theory by ensuring that the mother agrees exactly with all head feature specifications that the head daughters agree on, regardless of where the specifications come from.

Principles of UFI govern the well-formedness of the ID rule extension relation. They may be thought of as first applying to the ID rule output of metarule biclosure and then forever afterwards in RGPSG derivation. As we shall see, the lack of exceptional feature specifications in RGPSG means that ID rules that abrogate the constraints of UFI cannot be written.

Head feature convention. The head feature convention enforces the invariant that the mother is in absolute agreement with all head features on which the head daughters agree. It also requires the **BAR** value on a head daughter to be less than or equal to the **BAR** value on the mother. *HEAD* contains exactly those features that must be equivalent on the mother and head daughters of every ID rule.[12]

$$\begin{aligned} \textit{HEAD} \;=\; \{&\textbf{AGR}, \textbf{ADV}, \textbf{AUX}, \textbf{INV}, \textbf{LOC}, \textbf{N}, \textbf{NFORM}, \textbf{PAS}, \textbf{PAST}, \\ &\textbf{PER}, \textbf{PFORM}, \textbf{PLU}, \textbf{PRD}, \textbf{V}, \textbf{VFORM}\} \end{aligned}$$

Control agreement principle. The control agreement principle (CAP) differs from the HFC in that it establishes equivalences (*links*) between the categories in an ID rule: when two categories are *linked* in an ID rule, the two categories must be identical in any legal extension of that rule. Links are calculated immediately after the HFC has applied to the ID rules for the first time; once a link is established in an ID rule, it cannot be changed or undone.[13] The first part of the CAP calculates control relations between categories, while the second part of the CAP establishes links using the control relations. In all cases, linking is indicated by co-subscripting.

[12] In order to properly account for feature instantiation in the binary and iterating coordination schemata, the binary head (*BHEAD*) features **BAR**, **SUBJ**, **SUBCAT**, and **SLASH** are considered to be head features for the purposes of the HFC in all nonlexical, multiply-headed ID rules.

[13] In GKPS, only head feature specifications and inherited foot feature specifications determine the semantic types relevant to the definition of control. RGPSG simplifies this by considering inherited feature specifications and only some head feature specifications. Alternatively, control relations could be calculated every time the HFC instantiates a feature specification, although this would violate monotonicity. In fact, the RGPSG CAP uses links solely to preserve monotonicity of feature instantion in ID rule projection, which makes it easier to understand the consequences of a grammar.

$$CONTROL = \{\texttt{SLASH}, \texttt{AGR}\}$$

RGPSG control relations are calculated as follows. A *predicate* is a *VP* or an instantiation of *XP*[+PRD] such as a predicate nominal or adjective phrase. The *control feature* of a category C, where $C(\texttt{BAR}) \neq 0$, is SLASH if C is specified for SLASH; otherwise, it is AGR. Control is calculated once and for all immediately after the HFC has applied to the ID rules resulting from metarule biclosure.

Let f be the control feature of a category C_1. Then C_1 is controlled by C_2 in a rule if and only if $C_1(f) = C_2$, $C_2 \sqsupseteq X2$, and either the rule is $C_0 \rightarrow C_1 : C_2$ (recall that C_1 is the head daughter), or the rule is $C_0 \rightarrow C_3 : C_1, C_2$, and $C_0, C_1 \sqsupseteq VP$.

The RGPSG control agreement principle states: In an ID rule

$$C_0 \rightarrow C_1, \ldots, C_j : C_{j+1}, \ldots, C_n$$

- If C_i controls C_k and f_k is the control feature of C_k, then $C_k(f_k)$ and C_i are linked.
- If there is a nonhead predicate C_i with no controller, then link $C_i(f_i)$ and $C_0(f_0)$, where f_i and f_0 are the control features of C_i and C_0, respectively.

In the theory of GKPS, the control agreement principle performs subject-verb agreement by enforcing a control relation between the two daughters of the rule

$$S \rightarrow \texttt{H[-SUBJ]}, X2$$

In RGPSG, this rule must be stated as

$$S \rightarrow X2\,\texttt{[-SUBJ,AGR}\ X2\texttt{]} : X2$$

if we wish to enforce the control relation between the two daughters. Because control relations in RGPSG are static (never recalculated), this control relation exists even if, say, $X2 = PP$. Fortunately, verbs will only be specified for legal *X2* values in the lexicon, and therefore any "questionable" control relations involving an *X2* other than *NP* or *S* are ignored at the lexical insertion level.

Foot feature principle. The foot feature principle (FFP) requires any foot feature specification instantiated on a daughter category to also be instantiated on the mother. The value assigned to an instantiated foot feature is identical to every instantiation of the same foot feature on other daughter categories. The FFP ensures that (i) the existence of inherited foot features on any category of an ID rule blocks instantiation of those foot features on any other component category of the rule, and (ii) inherited foot features are equivalent across all component categories of the rule (this second condition may be too strong). Both conditions are designed to fix an error in the foot feature principle (FFP) of GKPS, which permits material to be topicalized from inside a topicalized constituent.[14]

$$FOOT = \{\texttt{RE}, \texttt{SLASH}, \texttt{WH}\}$$

Because the empty string can be dominated only by a category of the form α[NULL +, SLASH α] in RGPSG, the FFP tries to ensure that every gap will have a unique filler. Unfortunately, it is impossible to truly guarantee recoverability of deletions in RGPSG, because the FFP can only locally constrain the rule-to-tree projection, and not the ID rules or the parse trees themselves. This situation is unavoidable in the GPSG framework, simply because SLASH does not always mark the complete path between a gap and its filler in accepted GPSG analyses. The classic example is the GPSG analysis of subject dependencies, where an *S/NP* is reanalyzed as a *VP*, effectively deleting an *NP* gap in subject position. In GKPS, this operation is performed by slash termination metarule 2 (pp.160-162): [SLASH *NP*] only marks the path from the filler to the mother of the reanalyzed *VP*. Another example is the GKPS (pp. 150-152) analysis of missing-object constructions

[14]ID rule 10 introduces topicalization constructions in English:

$$S \rightarrow X, H/X \tag{10}$$

The control agreement principle (CAP) ensures that the two X categories in the rule agree with each other. It is possible to instantiate [SLASH X] on the S mother and X nonhead daughter without violating the GKPS CAP or FFP as in 11, provided all occurrences of X are identical:

$$S/X \rightarrow X/X, H/X \tag{11}$$

The structure 11 satisfies the GKPS CAP because the χ-features on the nonhead daughter agree with the slash category on the head daughter; the GKPS FFP is met because the foot feature specifications instantiated on the mother are also instantiated on a daughter. Revised GPSG prevents such extractions. The permissibility of the local tree 11 means that every topicalization structure is infinitely ambiguous in GKPS, because the X/X nonhead daughter can be terminated with ϵ. See appendix A.3 for more details.

such as *John is easy to please.* In missing-object constructions, [SLASH *NP*] only marks the path from the embedded *NP* object gap to the *V2*[INF]/*NP* dominating *to please*, failing to continue through the *AP easy to please* to the filler *John.* Many sweeping changes would be necessary before the FFP would be able to strictly enforce recoverability of deletions in RGPSG.

Definition: Free Features

A feature f is *free in* the ID rule $r = C_0 \rightarrow C_1, \ldots, C_j : C_{j+1}, \ldots, C_n$ iff $\forall i,\ 0 \leq i \leq n,\ f \notin \mathtt{DOM}(C_i)$.

The foot feature principle states:

1. When first applying to an ID rule r, if a foot feature f is not free in r, then instantiate [f noBind] on all categories in r that are not specified for f.
2. When SLASH is instantiated on the mother, instantiate it on all non-lexical head daughters.[15]
3. When a foot feature f is instantiated on a daughter, instantiate it on the mother.

[15]This condition springs from the necessity of accounting for certain parasitic gap facts according to the traditional GPSG analysis of clausal structure. The problem arises in sentences of the form

Kim wondered [$_S$ which authors [$_{S/NP}$ reviewers of ___ always detested ___]] (12)

where the parasitic gap is introduced by a binary nonlexical rule 13

$$S \rightarrow X2\,[\text{-SUBJ,AGR}\ X2] : X2 \tag{13}$$

rather than a ternary lexical rule like other parasitic gaps. Instantiating SLASH on the *X2* nonhead daughter must force the identical SLASH specification on the mother and head daughter. SLASH isn't a head feature in RGPSG, so there is no other way to accomplish this. A possible solution is to replace 13 with rules to introduce clauses with and without parasitic gaps:

$$S \rightarrow X2\,[\text{-SUBJ,AGR}\ X2] : X2$$
$$S/NP \rightarrow X2\,[\text{-SUBJ,AGR}\ X2]/NP : X2\,[\text{-NULL}]/NP$$

We would then need to ensure that AGR was transferred from the head daughter to the nonhead daughter by the CAP, despite the presence of the SLASH feature.

4. When a foot feature f is instantiated on a mother, instantiate it on one or more nonhead daughters.[16]

5.6 Marking Conventions

The sole explicit marking convention in RGPSG is the *simple default* (SD). Unlike FCRs and FSDs, SDs are constructive, easy to understand and computationally tractable. Each SD is applied to each category (and may be understood) independent of all other categories and RGPSG formal devices. SDs are applied in order to ID rules immediately after the initial application of principles of UFI.

An SD contains a predicate and a consequent. The consequent is a list of feature specifications. The predicate is a Boolean combination of truth-values and feature specifications such that if a category C bears or extends a given feature specification, that feature specification is true of C, else false. If the predicate is true of a given category C in a rule and the consequent includes only unbound and unlinked features, then the feature specifications listed in the consequent are instantiated on C. Each SD is applied simultaneously to every top-level category in every rule exactly once, in the order specified by the grammar. Consider the following SD:

SD 1: if [SUBCAT] *then* [BAR 0]

If the target category C in a ID rule is specified for the SUBCAT feature, but unspecified for the BAR feature, then the SD will force the feature specification [BAR 0] on C.

Given a list of simple defaults whose symbol size is p, and given a set of ID rules whose symbol size is n, the resultant set of ID rules can at most contain $\theta(n \cdot p)$ symbols (attained if every SD were true of every category and every category consisted of a lone symbol).

The elimination of FCRs does not appreciably reduce the descriptive elegance of RGPSG grammars: see appendix C for an RGPSG describing English, roughly equivalent in symbol count and descriptive adequacy to the English GPSG provided by GKPS. This is true because the constraints expressed by FCRs are, for the most part, already expressed in the way ID rules and metarules are written. Conflicts between FCRs and existing

[16] Note that this condition will only affect the foot features REL and WH, and never SLASH.

categories typically either indicate an error on the grammar writer's part, overgeneration of possible structures, or the lack of inviolable constraint in GPSG. These conflicts are less likely to occur in RGPSG because the RGPSG's formal devices are more constrained than those in GPSG. More generally, although the computational weakening of the formal theory has decreased the descriptive power available to the grammar writer, the linguistic strengthening of the formal model has increased the descriptive elegance of the theory as a whole: more is described in RGPSG by the formal theory than by the grammar writer. Simply put, it is harder to write *unnatural* grammars in RGPSG. But if the elimination of FCRs proves unacceptable, then the complex symbol rules of Chomsky (1965:79–83) may be used to specify possible syntactic categories. Complex symbol rules could increase descriptive elegance, linguistic universalism, and empirical coverage without causing intractability. For example, complex symbol rules can make the **NFORM**, **PFORM**, and **VFORM** features universally mutually exclusive.

5.7 Derivation and projection in RGPSG

To conclude, we must determine how the formal subsystems described above fit together, beginning by formally specifying the class of RGPSGs and the languages they generate. A subsequent section translates the GKPS analysis of topicalization, expletive pronouns, and parasitic gaps to the RGPSG formal system.

The set of ID rules R' resulting from metarule biclosure, UFI, and SD application generates the language of the RGPSG as follows. If R' contains a rule $A \rightarrow \gamma$ with an extension $A' \rightarrow \gamma'$ that satisfies all principles of UFI and is an LP-acceptable ordered production, then for any string of terminals α and nonterminals β, we write $\alpha A' \beta \Rightarrow \alpha \gamma' \beta$. This is a derivation step. The language of an RGPSG contains all terminal strings that can be derived, using the ID rules, from any extension of the distinguished start category. Let $\stackrel{*}{\Rightarrow}$ be the reflexive transitive closure of $\Rightarrow$. Then the language $L(G)$ generated by G is

$$L(G) = \{ x \mid x \in V_T^* \text{ and } \exists C \in K[(C \sqsupseteq \textbf{Start}) \wedge C \stackrel{*}{\Rightarrow} x]\}$$

5.8 Complexity of RGPSG Recognition

The universal recognition problem for RGPSG is NP-complete. Recall that a problem is NP-complete iff it is NP-hard and in $\mathcal{NP}$. Informally, RGPSG-Recognition is in $\mathcal{NP}$ because the restricted ID rule format (no null-transitions) ensures a polynomial bound on the length of the shortest derivation. In the worst case a branching ID rule can be converted to a non-branching ordered local tree in a derivation (this happens when all nonhead daughters are erased, either by a metarule or by the universal ID rule (8), leaving behind at least one head daughter, which can never be erased). Once a category is expanded in a derivation, it *must* be lexically realized in the derived string. Therefore a terminal string of length n can be derived with at most $p \cdot n$ productions in an RGPSG with p productions. Appendix B.1 contains a formal proof.

Unfortunately, it is difficult to use our conceptual typology of computation trees to establish the NP-hardness of RGPSG Recognition. Although an RGPSG parse tree may appear structurally equivalent to a polynomial depth pruned OR computation tree, we must be careful. As in the reduction for GPSG Recognition, RGPSG categories can encode nodes in the computation tree, and ID rules can represent the Turing machine OR transitions. But the extension relation in the RGPSG derivation step is governed by the the RGPSG head feature convention, which ensures that all heads dominated by a common head will have the same head features. This means we cannot use the HFC to transfer unaltered tape squares from a configuration to its successors, which blocks the most obvious reduction. The trick is to encode the *entire* pruned computation tree in an RGPSG category and use ID rules to enforce the next-move relation between subsets of the category—see the proof in appendix B.2. The actual reduction is so complicated, we loose the conceptual advantage of the uniform class of computation trees. Terminal ambiguity and nonlocal agreement (via universal feature instantiation) in RGPSG permit a considerably simpler reduction from Satisfiability, a known NP-complete problem, as shown in appendix B.1.

The restrictions motivated above have resulted in a substantial decrease in complexity from the EXP-POLY time hardness of GPSG-Recognition. In fact, only two sources of intractability remain in RGPSG — lexical ambiguity and nonlocal feature agreement (compare figures 4 and 7).

This decrease in complexity is significant from both theoretical and prac-

Syntactic Categories	◁ Unit feature closure
ID Rules	◁ At least one head daughter ◁ Mother & heads [NULL -] ◁ Bounded branching (4)
Metarules	◁ Cannot directly affect heads ◁ Biclosure
UFI	◁ No exceptional feature specifications (UFI preserves simple invariant) ◁ More restricted linguistically ◁ Monotonic
Marking Conventions (Simple Default)	◁ No disjunctive consequences ◁ Cannot interfere with UFI ◁ No FCRs

Figure 7: RGPSG Major Changes

tical perspectives. First, NP-complete problems typically have good average time algorithms and highly efficient near-optimal solution techniques, while EXP-POLY problems do not. Next, the fastest recognizer known for GPSGs can require double-exponential time in the worst case, while RGPSG has a simple exponential time recognizer. Finally, NP-complete problems have efficient witnesses, while EXP-POLY hard problems do not. This means that RGPSG parses can always be verified efficiently, while GPSG parses cannot, in general.

6 Linguistic Analysis of English in RGPSG

This section reproduces three of the more intricate linguistic analyses of GKPS in order to illustrate RGPSG's formalisms. To reproduce their comprehensive analysis of English in toto would be a disservice to that work and is beyond the scope of this paper. Instead, a RGPSG roughly equivalent in symbol count and descriptive adequacy to their GPSG for English may be found in appendix C; the reader should consult GKPS for the accompanying linguistic exposition. In all cases, co-subscripting indicates the linking performed by the CAP.

The RGPSG grammar for English serves to demonstrate the empirical adequacy of the restricted RGPSG formal system. RGPSG is empirically superior to GPSG not because its English grammar is better, but because it achieves descriptive adequacy within a vastly more restricted class of grammars.

6.1 Topicalization

The rule (14a) expands clauses and rule (14b) introduces unbounded dependency constructions (UDCs) in English.

$$\begin{array}{l} a.\ S \rightarrow X2\,\texttt{[SUBJ -,AGR}\ X2\texttt{]} : X2 \\ b.\ S \rightarrow X2\,\texttt{[SUBJ +,SLASH}\ X2\texttt{]} : X2 \end{array} \tag{14}$$

In both cases the *X2* nonhead daughter controls the head daughter, and the control agreement principle links the value of the head daughter's control feature with the *X2* daughter, creating the ID rules in (15).

$$\begin{array}{l} a.\ S \rightarrow VP\,\texttt{[AGR}\ X2_1\texttt{]} : X2_1 \\ b.\ S\,\texttt{[SLASH noBind]} \rightarrow S\,\texttt{[SLASH}\ X2_2\texttt{]} : X2\,\texttt{[SLASH noBind]}_2 \end{array} \tag{15}$$

In the following discussion, [3s] and [3p] abbreviate [PER 3,-PLU] and [PER 3,+PLU], respectively. Consider the topicalization structure in figure 8, taken from GKPS, p. 145.

Note that it is impossible to extract any constituent out of the *X2* daughter in (15b) because the foot feature principle has forced [SLASH noBind] on the *X2* daughter and its mother. This explains the unacceptability of (16) in RGPSG, which is permissible in the theory of GKPS (see figure 9 in

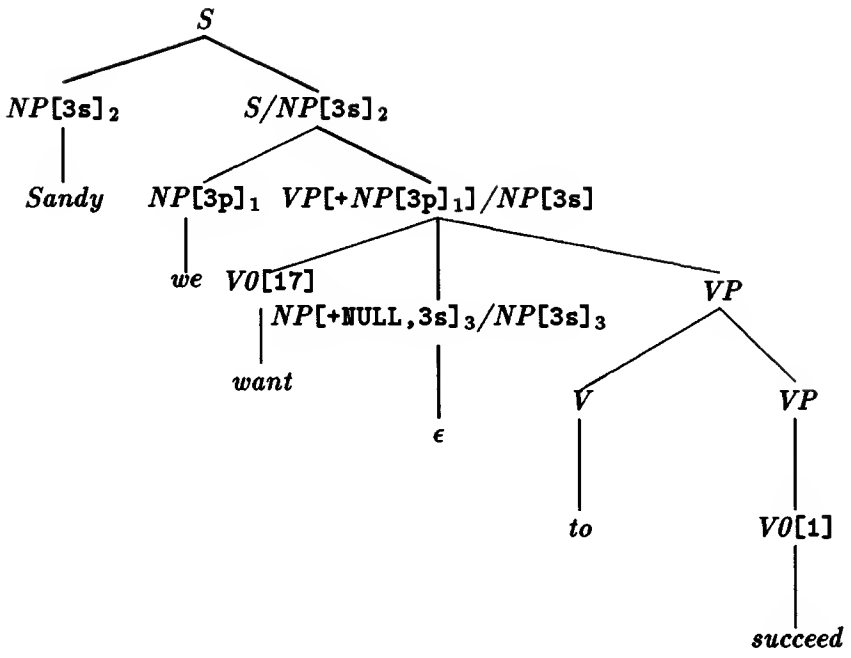


Figure 8: This is a typical topicalization structure in RGPSG. Co-subscripted categories are linked by the control agreement principle (a principle of universal feature instantiation), and therefore share all absent feature specifications. The foot feature principle and the CAP combine to ensure that all instances of the topicalized category (*NP*[3s]) agree. A dark line marks the extraction path.

appendix A.3).

* Boston [[a man from ___] [we want ___ to succeed]] (16)

6.2 Expletive pronouns

This section accounts for the distribution of the expletive pronouns *it* and *there* in infinitival constructions on the basis of postulated ID rules and principles of universal feature instantiation (see GKPS, pp.115-121). The feature specification [AGR *NP*[NFORM α]] is abbreviated as $+\alpha$ below, where α is it, there, or NORM.

The RGPSG for English includes the ID rules (17),

$$\begin{array}{l} a.\ S \rightarrow X2\,[\text{-SUBJ,AGR } X2] : X2 \\ b.\ VP \rightarrow [13] : VP[\text{INF}] \\ c.\ VP \rightarrow [16] : (PP[\text{to}]), VP[\text{INF}] \\ d.\ VP \rightarrow [17] : NP, VP[\text{INF}] \\ e.\ VP[\text{AGR } S] \rightarrow [20] : NP \end{array} \tag{17}$$

the simple defaults (18),

$$\begin{array}{l} a.\ SD\ 1{:}\ \textit{if}\ [\text{SUBCAT}]\ \textit{then}\ [\text{BAR 0}] \\ b.\ SD\ 2{:}\ \textit{if}\ [\text{+V,-N,-SUBJ}]\ \textit{then}\ [\text{+NORM}] \end{array} \tag{18}$$

the extraposition metarule (19),

$$\begin{array}{c} X2\,[\text{AGR } S] \rightarrow W \\ \Downarrow \\ X2\,[\text{+it}] \rightarrow W, S \end{array} \tag{19}$$

and the lexical entries (20). All other nouns are specified for [NFORM NORM] by their lexical entries.

$$\begin{array}{l} \langle it, NP[\text{PRO,-PLU,NFORM it}]\rangle \\ \langle there, NP[\text{PRO,NFORM there}]\rangle \end{array} \tag{20}$$

From the ID rules in (17), RGPSG generates the following ID rules.

$$\begin{array}{l} a.\ VP[\text{AGR}_1] \rightarrow V0[13,\text{AGR}_1] : VP[\text{INF,AGR}_1] \\ b.\ VP[\text{AGR}_1] \rightarrow V0[16,\text{AGR}_1] : (PP[\text{to}]), VP[\text{INF,AGR}_1] \end{array} \tag{21}$$

The absence of a controlling category allows the CAP to link the AGR values of the mother and *VP*[INF] predicate daughter. The HFC then links the AGR values of the mother and lexical head daughter. SD 1 specifies the head daughter for [BAR 0], while SD 2 cannot affect the linked AGR values.

$$VP[\text{AGR}_1\ NP[\text{NORM}]] \rightarrow V0[14, \text{AGR}_1\ NP[\text{NORM}]] : V2[\text{INF},\ \text{AGR}_1\ NP[\text{NORM}]]$$

The CAP and HFC operate identically as in (21), except that the [+NORM] specification is inherited from the ID rule (17b) and propagated through the rule by the CAP and HFC.

$$VP[\text{AGR}_2\ NP[\text{NORM}]] \rightarrow V0[17, \text{AGR}_2\ NP[\text{NORM}]] : NP_1, VP[\text{INF},\ \text{AGR}_1\ NP] \quad (22)$$

The *NP* daughter controls its *VP*[INF] sister, and the CAP links the AGR value of the *VP* to its sister *NP*. SD 2 specifies the mother for [+NORM], and the HFC forces this specification on the head daughter.

The rules in (23) introduce [+it] and [+there] specifications. Note that (23a) is the result of the extraposition metarule on the ID rule (17e).

$$\begin{array}{ll} a. & VP[\text{+it}] \rightarrow [20] : NP, S \\ b. & VP[\text{+it}] \rightarrow [21] : (PP[\text{to}]), S[\text{FIN}] \\ c. & VP[\text{AGR}\ NP[\text{+there,PLU}\ \alpha]] \rightarrow [22] : NP[\text{PLU}\ \alpha] \end{array} \quad (23)$$

The rules in (23) may only expand the *VP* daughters of the ID rules (21) and (22) in a derivation (compare their AGR values). Thus, the grammar claims that expletive pronouns only occur in utterances generated using the rules in (23), in combination with the "extending" rules (21) and (22). This describes the following facts from GKPS, p. 120.[17]

$$\left\{ \begin{array}{c} \text{It} \\ \text{*There} \\ \text{*Kim} \end{array} \right\} [\ \text{continues}\ [\ \text{to bother}\ [\ \text{Lou}\ ]\ [\ \text{that Robin was chosen}\ ]]] \quad (24)$$

$$\left\{ \begin{array}{c} \text{*It} \\ \text{There} \\ \text{*Kim} \end{array} \right\} [\ \text{appeared (to us)}\ [\ \text{to be}\ [\ \text{nothing in the park}\ ]]] \quad (25)$$

[17] In order to better understand these examples, associate each constituent with the ID rule that generated it. To help with this task, the main verbs and their SUBCAT values are: ⟨*continue*, *13*⟩, ⟨*appear*, *16*⟩, ⟨*believe*, *17*⟩, ⟨*bother*, *20*⟩, ⟨*be*, *22*⟩.

Leslie [believed { it / *there / *Kim } [to bother [us] [that Lee lied]]] (26)

We [believed { *it / there / *Kim } [to be [no flaws in the argument]]] (27)

6.3 Parasitic gaps

Simple parasitic gaps, that is, those introduced in verb phrases by lexical rules, present no problem for RGPSG because the FFP demands all instantiations of SLASH on daughters to be equal to each other and equal to the SLASH instantiation on the mother.

$$\begin{array}{l} VP/NP \\ \quad V0\,[13] \\ \quad NP/NP \\ \quad PP[\texttt{to}]/NP \end{array} \tag{28}$$

Kim wondered which models

Sandy { [had sent [pictures of ___] [to ___]] / [had sent [pictures of ___] [to Bill]] / [had sent [pictures of Bill] [to ___]] } (29)

The FFP insists nonlexical heads be instantiated for SLASH if any nonhead daughter is, thereby explaining the unacceptability of (30) and the acceptability of (31).

a. $*S/NP$
$\quad NP/NP$
$\quad VP$ (30)

b. * Kim wondered which authors
[[reviewers of ___] [always detested sushi]]

a. S/NP
$\quad NP/NP$
$\quad VP/NP$ (31)

b. Kim wondered which authors
[[reviewers of ___] [always detested ___]]

[redacted]

7 A Change in Perspective

This work is similar to that of Shieber (1986) in its attempt to reconstruct GPSG theory. Shieber, however, is concerned solely with creating a more easily implementable and understandable description of GPSG theory, rather than with changing the theory's generative or computational power.

A central goal of mathematical linguistics is to precisely determine the power of a linguistic theory. Traditionally, formal language theory (the Chomsky hierarchy) and its generative power analyses have translated this question into the narrower question of how unrestricted the rule format of a theory is. We have seen that modern computational complexity theory offers another, more useful, translation: how much of what computational resources does a theory consume? Complexity theory also offers a new perspective on descriptive adequacy. Descriptive adequacy, as commonly understood, refers to a theory's ability to assign the same structural descriptions to utterances that humans do. This is the perspective of formal language theory and E-language. But from a computational complexity perspective, descriptive adequacy refers to how faithfully the *internal* structure of a linguistic theory — its representations and internal operations — corresponds to the internal structure of our language facility. In a descriptively adequate linguistic theory, the structural descriptions *and* computational power of the theory match those of an ideal speaker-hearer.

8 References

Barton, E., 1985. On the complexity of ID/LP parsing. *Computational Linguistics* 11(4):205–218.

Barton, E., 1986. Constraint propagation in Kimmo systems. *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics*. Columbia University, New York: Association for Computational Linguistics, pp. 53–59.

Barton, E., R. Berwick, and E. Ristad, 1987. *Computational Complexity and Natural Language*. Cambridge, MA: MIT Press.

Berwick, R. and K. Wexler, 1982. Parsing efficiency and c-command. *Proceedings of the First West Coast Conference on Formal Linguistics*. Los Angeles, CA: University of California at Los Angeles, pp. 29–34.

Calder, J., E. Klein, and H. Zeevat, 1988. Unification Categorial Grammar: a concise, extendable grammar for natural language processing. In *Proceedings of the 12th International Conference on Computational Linguistics and the 24th Annual Meeting of the Association for Computational Linguistics*, Budapest, August, 1988.

Chandra, A., D. Kozen, and L. Stockmeyer, 1981. Alternation. *J. ACM* 28(1):114–133.

Chomsky, N., 1965. *Aspects of the Theory of Syntax*. Cambridge, MA: MIT Press

Chomsky, N., 1986. *Knowledge of Language: Its Origins, Nature, and Use*. New York: Praeger Publishers.

Evans, R., 1985. ProGram—a development tool for GPSG grammars. *Linguistics* 23(2):213-243.

Garey, M., and D. Johnson, 1979. *Computers and Intractability*. San Francisco: W.H.Freeman and Co.

Gazdar, G., 1982. Phrase structure grammar. In *The Nature of Syntactic Representation*, P. Jacobson and G. Pullum, eds. Dordrecht, Holland: Reidel.

Gazdar, G., E. Klein, G. Pullum, and I. Sag, 1985. *Generalized Phrase Structure Grammar*. Oxford, England: Basil Blackwell.

Gazdar, G., G. Pullum, R. Carpenter, E. Klein, T. Hukari, and R. Levine, 1988. Category Structures. *Computational Linguistics* 14:1–19.

Harrison, P. and M. Maxwell, 1986. A new implementation for Generalized Phrase Structure Grammar. *Proceedings of the 6th Annual Canadian Artificial Intelligence Conference.*

Kayne, R., 1981. Unambiguous paths. In *Levels of Syntactic Representation*, R. May and J. Koster, eds. Dordrecht: Foris Publications, pp. 143–183.

Lewis, H. and C. Papadimitriou, 1978. The efficiency of algorithms. *Scientific American* 238:96–109.

McCawley, J.D. 1982. Parentheticals and Discontinuous Constituent Structure. *Linguistic Inquiry* 13(1):91–106.

Pesetsky, D., 1982. Paths and categories. Ph.D. dissertation, MIT Department of Linguistics and Philosophy, Cambridge, MA.

Phillips, J. and H. Thompson, 1985. GPSGP—a parser of generalized phrase structure grammars. *Linguistics* 23(2):245–261.

Pollard, C., 1984. Generalized phrase structure grammars, head grammars, and natural language. Doctoral dissertation, Stanford University.

Pollard, C., 1985. Phrase Structure Grammar without Metarules. In WCCFL IV.

Ristad, E.S., 1986. Computational complexity of current GPSG theory. *Proceedings of the 24th Annual Meeting of the Association for Computational Linguistics.* Columbia University, New York: Association for Computational Linguistics, pp. 30–39.

Ristad, E. and R. Berwick, 1989. Computational consequences of agreement and ambiguity in natural language. *Journal of Mathematical Psychology*, 33(4):379–396.

Shieber, S., 1983. Direct parsing of ID/LP grammars. *Linguistics and Philosophy* 7(2):135–154.

Shieber, S., 1986. A simple reconstruction of GPSG. *Proceedings of the 11th International Conference on Computational Linguistics.* Bonn, West Germany, 20-22 August, 1986.

Zeevat, H., E. Klein, and J. Calder, 1987. Unification categorial grammar. In *Categorial Grammar, Unification Grammar, and Parsing*, Edinburgh

[redacted]

A Complexity of GPSG reconsidered

In this section the universal recognition problem for GPSGs is proved formally to be EXP-POLY time-hard, the number of local trees in the GKPS grammar for English is underestimated, and the consequences of GPSG's theoretical intractability are considered from the perspective of a grammar writer.

A.1 GPSG Recognition is EXP-POLY time-hard

This section provides a formal proof of the URP for GPSGs. It begins by providing a formal definition of alternating Turing machines, based on the definition in Chandra, Kozen, and Stockmeyer (1981). We have taken the work tapes to be one-way infinite instead of two-way infinite, in addition to making some other minor changes.

Definition. A k-tape alternating Turing machine is an 11-tuple:

$$M = < Q, \Sigma, \Gamma, \$, \#, k, \sigma, q_0, \mathit{Final}, U, E >$$

where

Q, q_0, Final	=	set of states, initial state, set of accepting states
Σ, Γ	=	input, tape alphabets, $\Sigma \subseteq \Gamma$
$\$, \#$	=	endmarker, blank symbol, $\$, \# \in \Gamma - \Sigma$
U	=	set of universal states, $U \subseteq Q$, U disjoint from *Final* and E
E	=	set of existential states, $E \subseteq Q$, E disjoint from *Final* and U
k	=	number of read-write tapes, $k \geq 1$
Q'	=	$U \cup E$
δ	=	next-move relation, where $\delta \subseteq (Q' \times \Gamma^{k+1}) \times (Q \times \Gamma^k \times \{\mathit{Left}, \mathit{Right}\}^{k+1})$
Left	=	-1
Right	=	$+1$

The ATM has a read-only input tape, with the input $w \in \Sigma^*$ written as $\$w\$$ and the reading head initialized to the first symbol of w. The k work tapes are one-way infinite and are initially blank. A *configuration* of the ATM consists of the state together with the head positions and contents of the

$k+1$ tapes. A move of the ATM consists of reading one symbol from the input tape and moving the heads left/right as allowed by δ, in addition to changing the state of the machine. The directions *Left* and *Right* have the numerical values $+1$ and -1 for convenience in proofs. δ does not include any transitions from accepting states. We say a configuration of M is *existential, universal,* or *accepting* if the state of the TM in that configuration is; in this formalization, an accepting configuration does not need any special tape contents, but only an accepting machine state.

For configurations C of M, let the sequence $Next_M(C) = (C_0, \ldots, C_{k-1})$ enumerate the possible successor configurations of C according to δ. k is bounded above by the number of pairs in the relation δ, which we may write as $|\delta|$.

The *computation* of an alternating TM M on an input w is a possibly infinite tree where the nodes correspond to ATM configurations, that is, an AND/OR computation tree whose outdegree is $|\delta|$. Each node of the computation contains a machine configuration that is reachable from the configuration above it, according to the next-move relation. However, to build a possibly infinite tree the nodes must be made mathematically distinct. This is accomplished by defining each node as a pair $\langle x, C\rangle$ where C is the machine configuration and x is a tree position. Technically, the tree position is a string of numbers that identify a position in the tree by listing which branch to take at each node; the numbers are all between 0 and $|\delta|-1$. The root position is the empty string, so the root node of the tree is $\langle \epsilon, C_0\rangle$ where C_0 is the initial configuration. The daughters of any node $\langle x, C\rangle$ are given by $NextNode_M(x, C)$ where

$$NextNode_M(x, C) = \{\langle xi, C_i\rangle : Next_M(C) = (\ldots, C_i, \ldots)\}.$$

The concatenation xi identifies a unique daughter of the position x by adding another branch number at the end.

The criterion for acceptance in an ATM computation tree is as follows. Let N be the set of nodes in the computation tree of ATM M on input w. We label the nodes of the tree either **true** or **false** as follows. A labeling $L : N \to \{\mathbf{true}, \mathbf{false}\}$ is said to be *acceptable* if the labeling of each node $\langle x, C\rangle$ satisfies the following conditions:

1. C is an accepting configuration and $L(x, C) = \mathbf{true}$.

2. C is an existential configuration and

$$L(x,C) = \bigvee_{\langle xi,C'\rangle \in NextNode_M(x,C)} L(xi,C')$$

3. C is a universal configuration and

$$L(x,C) = \bigwedge_{\langle xi,C'\rangle \in NextNode_M(x,C)} L(xi,C').$$

To simplify matters, we also require that $NextNode_M(x,C)$ be nonempty in this case.

By definition, $\bigvee$ of an empty set is **false**. M is defined to *accept* the input w if and only if $L(\epsilon, C_0) =$ **true** for all acceptable labelings L. Note that ATMs without universal states operate exactly as nondeterministic TMs do.

Chandra, Kozen, and Stockmeyer (1981) prove that

$$ASPACE(S(n)) = \bigcup_{c>0} DTIME(c^{S(n)})$$

where $ASPACE(S(n))$ is the class of problems solvable in space $S(n)$ on an ATM and $DTIME(F(n))$ is the class of problems solvable in time $F(n)$ on a deterministic Turing machine. In particular, when $f(n)$ is the class of all polynomial functions, the formula tells us that polynomial space on an ATM is equivalent to exponential-polynomial time on a deterministic TM.

The following proof of theorem 5 reduces instances of polynomial space-bounded alternating Turing machines to instances of GPSG Recognition.

Proof. By direct simulation of ATM M on input w.[18] Let M be a 1-tape ATM with polynomial space bound $S(n)$; let w be its input. Given these

[18] Without loss of generality, we use a 1-tape ATM, so

$$\delta \subseteq (Q' \times \Gamma \times \Gamma) \times (Q \times \Gamma \times \{Left, Right\} \times \{Left, Right\}).$$

Also, in the reduction, note that the word *input* refers to three completely distinct objects. The *ATM input string* w is the string which may or may not be in the language generated by the ATM. The *GPSG input string* x is the string which may or may not be in the language generated by the GPSG; x and w are *never* the same. The *reduction input* is the problem instance $\langle M, w\rangle$, *i.e.*, the ATM M and its input string w. It is important not to confuse the three distinct uses by believing, for example, that the GPSG accepts the same language as the ATM. They cannot accept the same language in principle.

reduction inputs, we will construct a GPSG G in polynomial time such that M accepts w iff

$$\$0w_1 1w_2 2 \ldots w_n(n)\$(n+1) \in L(G).$$

By Chandra–Kozen–Stockmeyer (1981), the class of problems solvable in polynomial space $S(n)$ on an ATM is exactly equivalent to the class of problems solvable in exponential polynomial time on a DTM. Therefore, given our following proof, we have the immediate result that GPSG-Recognition is $DTIME(c^{S(n)})$-hard, for all constants c, or EXP-POLY time-hard.

The basic plan of the reduction is to reproduce a *pruned computation tree* of the ATM as the *parse tree* of the GPSG. The GPSG will assign this elaborate structure to the empty string and not to the machine input. However, before the ATM simulation starts there will be some auxiliary structure that copies the machine input w into the features that represent the ATM input tape. The actual input that is presented to the grammar will therefore include an encoded version of w in addition to the "very long empty string" over which the computation tree is built.

Configurations of the ATM will be encoded as zero-level syntactic categories. Because the amount of tape the machine can use is bounded by the known quantity $S(|w|)$, we can use a separate feature to record the contents of each tape square. We also need three features to encode the ATM head position and current state. In a polynomial-time reduction, we are limited to specifying a polynomial number of features (for tape squares) and feature-values (for head positions), and that is why the reduction will be limited to polynomial space bounded ATM computations ($S(n)$ a polynomial).

The immediate domination (ID) rules of the GPSG will encode the δ relation of the ATM. The category corresponding to any configuration C can dominate the category corresponding to C' in a local tree iff δ licenses the transition $\langle C, C' \rangle$. (The exact details depend on whether the configuration C is universal or existential.) The reduction preserves the invariant that a nonterminal in the grammar can be terminated iff the configuration that it represents must be labeled **true** in the ATM computation. Consequently, the local tree for a universal configuration must include *every* successor configuration as a daughter. In contrast, the local tree for an existential configuration must merely include *some* successor configuration as a daughter. Nonterminals corresponding to halted, accepting configurations are terminated by the empty string.

Let $Next_M(C) = (C_0, \ldots, C_p)$. If C is a universal configuration, we would like to include the ID rule $C \rightarrow C_0, \ldots, C_p$; if C is an existential configuration, we would like to include $k+1$ rules of the form $C \rightarrow C_i$. However, we cannot use such rules directly in a polynomial-time reduction; there are far too many possible configurations, and we would need at least one rule for each. Instead, we must set up the features that encode the configurations in such a way that the ID rules only have to encode the finite δ relation, which is much smaller than the infinite $Next_M(\cdot)$ relation. Each δ-transition of the ATM licenses infinitely many transitions between configurations because δ does not care about the tape squares in a configuration that the machine is not currently scanning. In the same way, each ID rule of the constructed GPSG will project into a large number of local trees. The unchanged portion of a tape will not be transferred from a configuration to its successor by the ID rule, but will be transferred by the head feature convention (HFC, a principle of universal feature instantiation). All features that represent tape squares are declared to be head features and all daughters are head daughters. Consequently, the HFC will transfer the tape contents of the mother to the daughters except when prevented by the tape-writing activity specified by the next-move relation.

Proceeding to the details of the reduction, the following features are used to represent M-configurations:

STATE:	the state of the machine
INPUTPOS:	the head position of the read-only input tape
WORKPOS:	the head position of the read-write work tape
INPUT$_i$**:**	the contents of the i^{th} square of the input tape
WORK$_i$**:**	the contents of the i^{th} square of the work tape

In addition, the feature **PHASE** will be used to separate functionally distinct regions of the parse tree. **[PHASE READ]** categories are involved in reading the input string, **[PHASE RUN]** categories participate in the direct ATM simulation, and the **[PHASE START]** category links the **READ** and **RUN** phases.

As we have mentioned, the input string that is presented to the GPSG has the form

$$\$0w_1 1w_2 2 \ldots w_n(n)\$(n+1)$$

where the w_i are the characters of the machine input, the "$" characters are endmarkers, and $0, \ldots, (n+1)$ are regarded as additional characters. We must copy $\$w\$$ onto the input tape of the simulated machine. For every

character index i, $1 \leq i \leq |w|$, and for every possible character $a \in \Sigma$, include the following lexical rule for the lexical item ai:

$$\langle ai, \{\texttt{[PHASE READ]}, \texttt{[INPUT}_i\ a\texttt{]}\}\rangle$$

In addition, for every index over a wider range, $0 \leq i \leq |w| + 1$, include this lexical rule for the endmarker:

$$\langle \$i, \{\texttt{[PHASE READ]}, \texttt{[INPUT}_i\ \$\texttt{]}\}\rangle$$

Once these rules are constructed, they will work for other inputs w' of length $|w'| \leq |w|$ as well as for w. That is why endmarkers in the middle of w have been allowed in the copying rules.

Together with the specially formatted grammar input, these rules set up the input tape of the simulated ATM. We must also initialize the features that encode the work tape contents, the machine state, and the tape head positions. The initialization is completed by defining the distinquished start category `START` correctly:

$$\begin{aligned}\texttt{START} \;=\; & \{\texttt{[INPUTPOS 1]}, \texttt{[WORKPOS 1]}\} \\ & \cup\{\texttt{[STATE } q_0\texttt{]}, \texttt{[PHASE START]}\} \\ & \cup\{\texttt{[WORK}_j\ \texttt{\#]} : 1 \leq j \leq S(|w|)\}\end{aligned}$$

The following two ID rules are used to join the two subtrees together:

$$\begin{aligned}&\texttt{START} \rightarrow \{\texttt{[PHASE RUN]}\}, \{\texttt{[PHASE READ]}\} \\ &\{\texttt{[PHASE READ]}\} \rightarrow \{\texttt{[PHASE READ]}, \texttt{[PHASE READ]}\}\end{aligned}$$

(Here all daughters are head daughters.) The `[PHASE READ]` rule allows the input-reading portion of the tree to branch as many times as necessary to cover the input characters.

In our formal model of ATMs, the machine halts and accepts if it ever enters an accepting state $q \in \mathit{Final}$. Thus, for every such state we need a null-transition ID rule that will terminate the simulated computation tree. For every $q \in \mathit{Final}$, the following ID rule should be included:

$$\{\texttt{[STATE } q\texttt{]}, \texttt{[PHASE RUN]}\} \rightarrow \epsilon$$

However, the most important ID rules are still to come; they encode the next-move relation δ of the machine. Recall that

$$\delta \subseteq (Q' \times \Gamma \times \Gamma) \times (Q \times \Gamma \times \{\mathit{Left}, \mathit{Right}\} \times \{\mathit{Left}, \mathit{Right}\})$$

is a relation between tuples

$$\langle \text{state, input tape symbol, work tape symbol} \rangle$$

on the one hand and tuples

$$\langle \text{new state, new work tape symbol,} \\ \text{input head movement, work head movement} \rangle$$

on the other hand. We describe δ as

$$\delta(q,a,b) = \{\langle q',b',d_I,d_W\rangle : \\ \langle\langle q,a,b\rangle,\langle q',b',d_I,d_W\rangle\rangle \in \delta\}.$$

With this notation, we may specify the ID rules that encode δ. A set of rules will be specified for every state $q \in Q'$ and all tape symbols a and b, thus covering all of δ. No rules are to be constructed when $\delta(q,a,b) = \emptyset$. For $q \in Q'$ with $\delta(q,a,b) \neq \emptyset$ there are two cases depending on whether q is existential or universal. In either case, the construction must be carried out for all possible input-head positions i ($0 \leq i \leq |w|+1$) and work-head positions j ($1 \leq j \leq S(|w|)$):

1. If q is in E (an existential state), include an instance of the following ID rule for every $\langle q',b',d_I,d_W\rangle \in \delta(q,a,b)$:

$$\{\texttt{[INPUTPOS } i\texttt{]}, \texttt{[INPUT}_i\ a\texttt{]}, \\ \texttt{[WORKPOS } j\texttt{]}, \texttt{[WORK}_j\ b\texttt{]}, \\ \texttt{[STATE } q\texttt{]}, \texttt{[PHASE RUN]}\} \rightarrow \\ \{\texttt{[INPUTPOS } i+d_I\texttt{]}, \texttt{[INPUT}_i\ a\texttt{]}, \\ \texttt{[WORKPOS } j+d_W\texttt{]}, \texttt{[WORK}_j\ b'\texttt{]}, \\ \texttt{[STATE } q'\texttt{]}, \texttt{[PHASE RUN]}\}$$

   Each of these rules propagates the value on the input tape, changes the value on the work tape, moves the heads, and changes the automaton state; note that all have the same left-hand side. Because several such rules are included, only one daughter computation has to succeed. The lone daughter in each such rule is a head daughter.

2. If q was not in E, it must be in U instead (a universal state). For this case, let $L \rightarrow R_1, \ldots, L \rightarrow R_p$ be the rules that would have been constructed according to case (a) if q had been an existential state. Then include the rule

$$L \rightarrow R_1, \ldots, R_p$$

   instead of those rules. Again, every daughter is a head daughter.

With these rules, the construction is almost finished; only a few loose ends remain. The syntactic categories used in our GPSG are formally specified as follows:

$$
\begin{aligned}
\mathit{Feat} &= \{\texttt{STATE}, \texttt{INPUTPOS}, \texttt{WORKPOS}, \texttt{PHASE}\} \\
&\quad \cup \{\texttt{INPUT}_i : 0 \le i \le |w|+1\} \\
&\quad \cup \{\texttt{WORK}_j : 1 \le j \le S(|w|)\} \\
\texttt{Atom} &= \mathit{Feat} \\
\rho(f) &= \begin{cases} Q, \text{ if } f = \texttt{STATE} \\ \text{the set } \{i : 0 \le i \le |w|+1\}, \text{ if } f = \texttt{INPUTPOS} \\ \text{the set } \{j : j \le j \le S(|w|)\}, \text{ if } f = \texttt{WORKPOS} \\ \Sigma \cup \{\$\}, \text{ if } f = \texttt{INPUT}_i \text{ for some } i \\ \text{the ATM tape alphabet } \Gamma, \text{ if } f = \texttt{WORK}_j \text{ for some } j \\ \text{the set } \{\texttt{START}, \texttt{READ}, \texttt{RUN}\}, \text{ if } f = \texttt{PHASE} \end{cases}
\end{aligned}
$$

The set of head features is defined to consist of the $\texttt{INPUT}_i$ features and the $\texttt{WORK}_j$ features. In addition, we need feature co-occurrence restrictions to ensure full specification of all non-null categories. For every $f \in \texttt{Atom}$, include the FCR $[\texttt{STATE}] \supset [f]$.

Inspection of the construction steps shows that the reduction may be performed in polynomial time in the size of the simulated ATM. (Note that the grammar we construct encodes only the *description* of the machine that produces the computation tree—not the potentially infinite computation tree itself.)

No metarules or LP statements are needed, although metarules could have been used instead of the head feature convention. Both devices are capable of transferring the contents of the ATM tape from the mother to the daughter(s). One metarule would be needed for each tape square/tape symbol combination in the ATM.

GKPS definition 5.14 of admissibility (p.104) guarantees that admissible trees must be terminated. By the construction above, a [PHASE RUN] node can be terminated only if it represents an accepting configuration. In particular, a [PHASE RUN] node cannot be terminated by a lexical rule, because all constructed lexical rules are [PHASE READ]. This means the only admissible trees are accepting ones whose yield is the input string followed by a very long empty string. □

A.2 Number of CF English Productions

For GPSG and all its variants, the only grammar *directly* usable by the Earley algorithm, that is, with the same complexity as a context-free grammar, is the set of admissible local trees. I estimate the number of local trees in the following "typical" RGPSG for English and show, in accordance with earlier estimates (see Shieber 1983:137), that this set is astronomical. Any recognition procedure that explicitly calculates or uses the set of admissible local trees can only result in a slower recognition time than one that does not.

Consider the simplest ID rule 32 in the RGPSG for English.

$$VP \rightarrow [1]: \tag{32}$$

The VP mother may receive multiple values (or remain unspecified) for the atomic-valued features **CASE**, **GER**, **NEG**, **POSS**, **REMOR**, **WHMOR**, **AUX**, **INV**, **LOC**, **PAST**, **PER**, **PLU**, **PRD**, or **VFORM**. Assume that each feature is binary. Then 3^{14} possible extensions of the VP are licensed, since each feature may be $+$, $-$, or unspecified. VP may also receive many **AGR** specifications in which the atomic-valued features **CASE**, **COMP**, **GER**, **NEG**, α**FORM**, **POSS**, **REMOR**, **WHMOR**, **ADV**, **AUX**, **INV**, **LOC**, **N**, **PAST**, **PER**, **PLU**, **SUBJ**, **V** may receive multiple values or be undefined. The daughter's feature values are fixed by the lexicon and the HFC, so the ID rule (32) corresponds to $3^{14} \cdot 3^{19} = 3^{33}$ unanalyzable context-free productions. The GPSG equivalent of (32) corresponds to significantly more context-free productions due to the combinatorial possibilities of embedded categories in GPSG.

The ID rule (33) is slightly more complicated.

$$VP \rightarrow [2]: NP \tag{33}$$

The VP mother in (33) may bear all of the features of the VP mother in the rule 32, plus it may also bear the category-valued features **SLASH** or **WH**, or **RE**, because these foot features can be instantiated on the NP daughter. ID rule (33) therefore corresponds to approximately $3^{14} \cdot (3^{19})^3 = 3^{71} > 10^{33}$ unanalyzable context-free productions. We would expect that only some of these 10^{33} context-free productions are really legitimate rules of an English RGPSG. Even if we were able to exclude the invalid extensions from consideration, the RGPSG for English would still contain an astronomical number of context-free productions, and the GPSG for English still more.

Significantly underspecified ID rules such as the binary coordination schema correspond to an even greater number of context-free productions. In the following estimate, I count the three mutually-exclusive atomic head features **NFORM**, **PFORM**, **VFORM** as one feature, and ignore the features **NULL**, **CONJ**, **COMP** since their distribution is extremely limited. I must also ignore the positive Kleene star categories of the iterating coordination schema, because any ID rule containing them corresponds to an infinite number of context-free productions.

The 12 atomic head features can receive any value on either head daughter ($= (3^{12})^2$) and the 9 non-head atomic features can receive any value on any of the three categories in the rule ($= (3^9)^3$). Because the foot features **WH** and **SLASH** are mutually exclusive, there are effectively 3 category-valued features: the head feature **AGR** may take 3^{12+9} values on either head daughter ($= (3^{21})^2$), while the two foot feature may each take 3^{21} possible values on only one category ($= (3^{21})^2$). Thus, the BCS corresponds to

$$(3^{12})^2 \cdot (3^9)^3 \cdot (3^{21})^2 \cdot (3^{21})^2 = 3^{135} \geq 10^{64}$$

context-free rules. In short, even the more constrained RGPSG framework licenses an astronomical number of context-free productions.

A.3 Practical Consequences of GPSG's Complexity

Here I argue that the GPSG theory of GKPS is difficult to understand and prone to massive overgeneration. The exclusive use of extensional (i.e. nonconstructive) definitions, when coupled with the vast array of extensional possibilities and incompletely specified relationships among formal devices means that knowledge of principles does not translate into an ability to determine the consequences of that knowledge. It is difficult, if not impossible, for the linguist to understand the consequences of a particular GPSG system, in part because the formal system is so intractable.

Not only are there an extra-astronomical number of syntactic categories in GPSG, but the computations performed on them are extremely intricate. Head features are, in principle, those features that must agree on the mother and head daughters in a local tree. Consider the head features **SUBJ**, **SUBCAT**, **SLASH**, and **BAR**, which are actually required to *disagree* in nearly all cases. **SUBCAT**, for example, must never be equivalently specified on the mother and head daughters, except when it remains unspecified on both.

ID rules with lexical heads predominate, and in those rules BAR, SUBJ, and SLASH features must never agree. The GKPS solution to this problem is to make feature instantiation operate only on those (non-problematic) features that may be freely equated: absolute feature specification identity is not possible to enforce, either "because the 'problematic' feature specification is stipulated in the rule, or because its presence or absence is required by the FCRs, or because its presence or absence is required by the FFP or CAP."(p.95) In short, principles of universal feature instantiation can only be understood relative to all other formal devices. The dynamic nature of feature instantiation, when compounded with extensional (i.e. non-constructive) definitions of the foot feature principle (FFP), control agreement principle (CAP), and head feature convention (HFC), results in a linguistic theory that is extremely difficult to understand.

As shown above, the 'problematic feature specification' solution introduces significant additional complexity in the theory of syntactic categories: universal feature instantiation plays a central role in the EXP-POLY time-hard reduction. The additional complexity is concealed by extensional definitions — admissible local trees are defined in terms of all projections of an ID rule that meet the FFP, CAP, and HFC. The interactions of these three grammatical components remain unspecified, and the actual role of FCRs is similarly never explained. If FCRs apply after the HFC, all lexical heads would first be specified for [BAR 2] and then eliminated; if FCRs apply before the HFC, all lexical heads would be specified for [BAR 2] and then admitted. The GKPS solution to this dilemma is to apply FCRs and the HFC *simultaneously* to the ID rule projections, along with the FSDs and other principles of universal feature instantiation.

One might think from these extensional definitions that the FFP, CAP, HFC, FCRs, and FSDs apply relatively independently. Even worse, one might associate order of presentation (FFP, then CAP, then HFC, and finally FSDs) with formal dependence. In fact, their interactions are intricate and highly structured. For example, the CAP depends heavily on the HFC because head feature specifications in part determine the semantic types relevant to the definition of control. One of the goals of Revised GPSG, as presented here, is to unravel these dependencies and expose the underlying structure of feature instantiation in natural language grammars.

A major caveat regarding the following examples is in order. The major empirical support for the claim that the interactions among GPSG's formal

devices are unpredictable is the theory's computational intractability, as shown above in section 4. I in no way consider errors in GKPS to be a major source of empirical evidence for this claim. The errors are anecdotal, circumstantial, and probably unavoidable in any system as complex as a descriptively adequate linguistic theory — at best these errors constitute auxiliary support for this claim, and for this reason appear in the appendix.

These interactions manifest themselves in extremely subtle ways. A case in point is the GKPS analysis of unbounded dependency constructions (UDC's). These constructions are said to have a top, middle, and bottom. The top of topicalization and wh-movement constructions is supplied by the ID rule (34),

$$S \rightarrow X2, H/X2 \tag{34}$$

which is repeatedly assumed in GKPS—throughout chapter 5 and in the elucidation of the CAP, e.g. local tree (25c) on p.90—to result in local trees of the form

S
    NP[PER 3, -PLU]
    S[SLASH *NP*[PER 3, -PLU]]

where the CAP requires the value of SLASH on the head to be identical to the controller *X2* (that is, the two *X2* categories of the ID rule (34) must agree). This local tree is an egregious violation of the HFC, because the non-problematic head feature SLASH is present on the head daughter but not the mother.

A tree compatible with the HFC, CAP, and FFP follows immediately. Note that this is the most general tree, because the FFP requires identical instantiation of SLASH values on the mother and daughters.

S[SLASH *NP*[PER 3, -PLU]]
    NP[PER 3, -PLU, SLASH *NP*[PER 3, -PLU]]
    S[SLASH *NP*[PER 3, -PLU]]

An immediate consequence of the HFC, then, is that UDC constructions are topless, consisting of an infinite iteration of NP's missing NP's internally

(FSD 3 prevents +NULL from being instantiated on the NP/NP daughter). A UDC cannot terminate without introducing another UDC. The obvious solution is to remove SLASH from the set of head features — it was unclear why SLASH should be a head feature from the start, since most heads are prevented from being specified with SLASH by FCR 6 (and by FSD 3, indirectly through FCR 19). The strongest argument for including SLASH in HEAD arises from the GKPS account of parasitic gap facts. Removing SLASH from HEAD also simplifies the definition of the control relation to only consider HEAD and inherited FOOT feature specifications, rather than non-FOOT HEAD features and inherited FOOT features, when determining the semantic types relevant to the definition of control (see GKPS p.87).

Unfortunately, our troubles are not over yet. It is perfectly admissible to *instantiate* a SLASH feature on the mother, provided we satisfy the FFP and instantiate it on at least one daughter. Since SLASH cannot be instantiated twice on the daughter head, it must be instantiated identically on the daughter NP.

S[SLASH *NP*[PER 3, -PLU]]
    NP[PER 3, -PLU, SLASH *NP*[PER 3, -PLU]]
    S[SLASH *NP[PER 3, -PLU]*]

The CAP is satisfied: the value of the SLASH feature on the agreement target (i.e. the head daughter) is identical to the χ-specifications of the controller unified with the value of the SLASH feature on the controller. (In plainer English, the χ-specifications are the head but not foot specifications, plus any inherited foot specifications. In this instance, the χ-specifications of the controller are the features {[+N,-V,BAR 2,PER 3,-PLU]}.)

In conclusion, the only permissible local tree in the entire preceding discussion is:

S[SLASH *NP*[PER 3, -PLU]]
    NP[PER 3, -PLU, SLASH *NP*[PER 3, -PLU]]
    S[SLASH *NP*[PER 3, -PLU]]

Therefore, even if we remove SLASH from the set of head features, the GKPS grammar for English will allow an infinite class of ungrammatical utterances, such as that shown in figure (9).

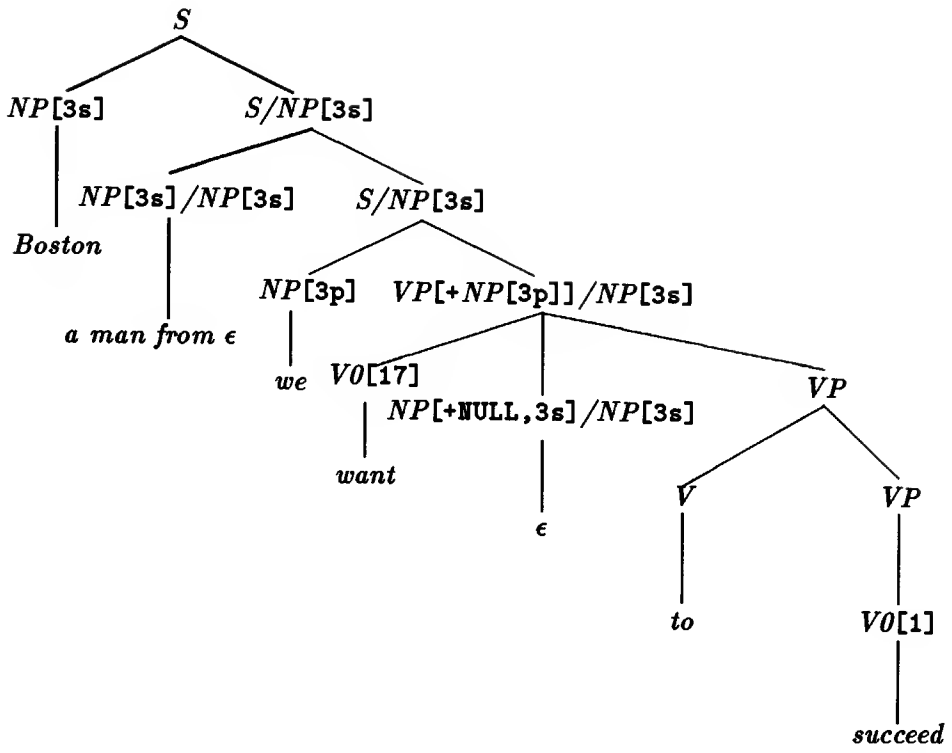


Figure 9: This is a problematic topicalization structure in GPSG, where [SLASH *NP*[3s]] has been instantiated on both the mother *S* and its *NP*[3s] nonhead daughter, in accordance with the FFP. The bad extraction is marked by a dark line.

Other examples of unexpected interactions among the principles of universal feature instantiation can be found in GKPS. An example is the definition of the FFP given in GKPS. Consider the ID rules generated by the STM2 metarule, which are of the form:

$$A/B \rightarrow C, D$$

Local trees of the class

A/B
    C/F
    D/G

meet the FFP because the unification of instantiated foot features on the daughter categories is undefined for **SLASH** (the unification of F and G is undefined), and $\phi(A/B) \mid \text{FOOT} \sim A/B$ is also undefined because $\text{FOOT} \sim A/B$ is the empty set.

The RGPSG FFP fixes this problem as follows. Any foot feature specification that is instantiated on a daughter category in an RGPSG local tree must also be instantiated in the mother category, and that specification must be identical to an instantiation of the same feature on other daughter categories. This revised FFP also ensures that inherited foot features on the mother prevent an instantiation of those foot features on any daughters.

B Complexity of RGPSG Recognition

This appendix contains a formal proof that the universal recognition problem for R-GPSG's is NP-complete. That is, the problem of determining for an arbitrary RGPSG G and input string w whether w is in the language $L(G)$ generated by G, is NP-complete.

B.1 RGPSG Recognition is NP-complete

Let P be the set of ID rules resulting from applying metarule biclosure, UFI, and simple defaults to the set of ID rules given in the RGPSG G. Recall that P contains $O(|G|^5)$ symbols.

Lemma B.1 *Let $(\varphi_0, \ldots, \varphi_k)$ be a shortest leftmost derivation of φ_k from φ_0 in an RGPSG G containing at least one branching production.*[19] *If $k > |P|$, then $|\varphi_k| > |\varphi_0|$.*

Proof. In the derivation step $\varphi_i \Rightarrow \varphi_{i+1}$, where $\varphi_i = \alpha A' \beta$ and $\varphi_{i+1} = \alpha \gamma' \beta$ for $\alpha \in V_T^*$, $\beta \in (V_T \cup K)^*$, one of the following cases must hold:

1. The production $A \rightarrow \gamma$ with extension $A' \rightarrow \gamma'$ is nonbranching ($|\gamma| = 1$). In the worst case, we could cycle through every possible nonbranching production (without using a branching production), after which we would begin to reuse them. Any extension of a production that has already been used in this run of nonbranching productions could have been guessed previously, and the length of the shortest nonbranching run must be less than $|P|$.

2. The production $A \rightarrow \gamma$ with extension $A' \rightarrow \gamma'$ is branching ($|\gamma| > 1$). Then $|\varphi_i| > |\varphi_{i+1}|$.

A total of at most $n-1$ branching productions derives an utterance of length n, because there are no null-transitions in an RGPSG.[20] Each branching

[19] If the RGPSG G does not contain a branching production, then $L(G)$ contains only strings of length one and all shortest derivations are shorter than $|P|$: membership for such a grammar is clearly in $\mathcal{NP}$.

[20] The only null-transition is the lexical element for the category $X2[+\texttt{NULL}]_1/X2_1$, as shown in 8. Null-transitions may, at the worst, convert the extension of a branching

production can be separated from the closest other branching production in the derivation by a run of at most $|P|$ nonbranching productions, and the shortest derivation of x will be of length $\theta(|P| \cdot |x|) = \theta(|G|^5 \cdot |x|)$. □

Theorem 6 *RGPSG Recognition is in $\mathcal{NP}$.*

Proof. On input RGPSG G and input string $x \in {V_T}^*$, guess a derivation of x in nondeterministic polynomial time as follows.[21]

1. Compute the set P of ID rules resulting from applying the simple defaults (while respecting the principles of UFI) to the output of metarule biclosure *BC(M,R)*. This can be done in deterministic polynomial time (see above).
2. Guess an extension S' of the start category S, and let S' be the derivation string.
3. For a derivation string $\alpha A' \beta$, where $\alpha \in V_T^*, \beta \in (V_T \cup K)^*$, guess a production $A \rightarrow \gamma$ and extension $A' \rightarrow \gamma'$ of it. Let $\alpha\gamma'\beta$ be the new derivation string.
4. If $\alpha\gamma'\beta = x$, accept.
5. If $|\alpha\gamma'\beta| > |x|$, reject.
6. Loop to step 3 (at most $|P| \cdot |x|$ times).

Every loop of the nondeterministic algorithm performs one step in the derivation. By lemma B.1, the shortest derivation of x is at most of length $\theta(|P| \cdot |x|)$, so we need to loop through the algorithm at most that many times. Guessing an extension of a category may be performed in time $\theta(|\texttt{Cat}| \cdot |\texttt{Atom}|)$, and an extension of a production may be guessed in time $\theta(|\texttt{Cat}| \cdot |\texttt{Atom}| \cdot |P|)$. This nondeterministic algorithm runs in polynomial time and accepts exactly $L(G)$; hence RGPSG Recognition is in $\mathcal{NP}$. □

The idea of the following hardness proof arose during a discussion with Ed Barton and Robert Berwick.

---

production to a nonbranching one in the derivation because no head daughter may bear the [+NULL] specification, and therefore null-transitions may in effect be compiled out of the derivation when we choose an ordered extension to expand the nonterminal A'.

[21] Again, we assume P contains at least one branching production. If not, then we should only loop as many times as there are productions, and then halt.

Theorem 7 *RGPSG Recognition is NP-hard.*

Proof. We reduce 3SAT to RGPSG Recognition in polynomial time. Given a 3CNF formula f of length m using the n variables $q_1 \ldots q_n$, we construct an RGPSG G_f such that the string w is an element of $L(G_f)$ iff f is satisfiable, where w is the string of formula literals in f. G_f is constructed as follows:

1. G_f includes the set `Atom` of atomic feature $\{\texttt{STAGE}, \texttt{LITERAL}, q_1, \ldots, q_n\}$ with values defined by the function ρ:

$$\begin{array}{lcl} \rho(\texttt{STAGE}) & = & \{1, \ldots, n+3\} \\ \rho(\texttt{LITERAL}) & = & \{\texttt{+}, \texttt{-}\} \\ \rho(q_i) & = & \{\texttt{0}, \texttt{1}\} \end{array}$$

   The set of head features HEAD is $\{q_1, q_2, \ldots, q_n\}$. The grammar will assign truth-values to the variables and check satisfaction in $n+3$ stages as synchronized by the feature `STAGE`. The start category is $\{[\texttt{STAGE 1}]\}$.

2. At each of the first n stages, a value is chosen for one variable; because the q_i are head features, the values that are chosen will be maintained throughout the derivation tree by the HFC. The following $2n$ non-branching rules are needed, constructed for all i, $1 \leq i \leq n$. All daughters are heads.

$$\begin{array}{lcl} \{[\texttt{STAGE } i], [q_i \texttt{ 0}]\} & \rightarrow & \{[\texttt{STAGE } i+1], [q_i \texttt{ 0}]\}: \\ \{[\texttt{STAGE } i], [q_i \texttt{ 1}]\} & \rightarrow & \{[\texttt{STAGE } i+1], [q_i \texttt{ 1}]\}: \end{array}$$

3. At stage $n+1$, the grammar has guessed truth assignments for all variables; all that remains is to use the truth assignments to generate satisfied three-literal clauses. The following two rules generate enough clauses to match the number of clauses in w:

$$\begin{array}{lcl} \{[\texttt{STAGE } n+1]\} & \rightarrow & \{[\texttt{STAGE } n+2]\}: \\ \{[\texttt{STAGE } n+1]\} & \rightarrow & \{[\texttt{STAGE } n+1]\}\,\{[\texttt{STAGE } n+2]\}: \end{array}$$

4. At stage $n+2$, the grammar generates satisfied three-literal clauses—clauses containing at least one true literal. Let C_0 and C_1 be the following categories:

$$\begin{array}{lcl} C_0 & = & \{[\texttt{STAGE } n+3], [\texttt{LITERAL -}]\} \\ C_1 & = & \{[\texttt{STAGE } n+3], [\texttt{LITERAL +}]\} \end{array}$$

Then the following 7 ternary-branching rules are needed; any set of three literals makes the clause true, provided at least one literal is true:

$$\begin{aligned}
\{\texttt{[STAGE } n+2\texttt{]}\} &\rightarrow C_0\ C_0\ C_1: \\
\{\texttt{[STAGE } n+2\texttt{]}\} &\rightarrow C_0\ C_1\ C_0: \\
\{\texttt{[STAGE } n+2\texttt{]}\} &\rightarrow C_1\ C_0\ C_0: \\
\{\texttt{[STAGE } n+2\texttt{]}\} &\rightarrow C_0\ C_1\ C_1: \\
\{\texttt{[STAGE } n+2\texttt{]}\} &\rightarrow C_1\ C_0\ C_1: \\
\{\texttt{[STAGE } n+2\texttt{]}\} &\rightarrow C_1\ C_1\ C_0: \\
\{\texttt{[STAGE } n+2\texttt{]}\} &\rightarrow C_1\ C_1\ C_1:
\end{aligned}$$

5. Finally, lexical insertion at stage $n+3$ ties together the truth-values chosen for the variables and the literals. For every q_i, $1 \leq i \leq n$, we need the following four lexical entries, bringing us to a total of $6n+9$ rules:

$$\begin{aligned}
&\langle q_i, \{\texttt{[STAGE } n+3\texttt{]}, \texttt{[LITERAL +]}, [q_i\ 1]\}\rangle \\
&\langle q_i, \{\texttt{[STAGE } n+3\texttt{]}, \texttt{[LITERAL -]}, [q_i\ 0]\}\rangle \\
&\langle \overline{q}_i, \{\texttt{[STAGE } n+3\texttt{]}, \texttt{[LITERAL +]}, [q_i\ 0]\}\rangle \\
&\langle \overline{q}_i, \{\texttt{[STAGE } n+3\texttt{]}, \texttt{[LITERAL -]}, [q_i\ 1]\}\rangle
\end{aligned}$$

There are no category-valued features, LP statements, metarules, or simple defaults in the RGPSG G_f constructed by the reduction.

If some extension of the start category $S = \{\texttt{[STAGE 1]}\}$ can be generated, then the formula f is satisfiable; each extension of the start category that generates a string must encode a satisfying truth assignment. For example, the category

$$\{\texttt{[STAGE 1]}, [q_1\ 1], [q_2\ 0], \ldots, [q_n\ 1]\}$$

generates 3-CNF formulas f with the satisfying truth assignment $q_1 = 1, q_2 = 0, \ldots, q_n = 1$. Note that the RGPSG constructed in the reduction generates *all* satisfiable 3CNF Boolean formulas, of any length, using n or fewer variables. □

B.2 Computation Tree Reduction for RGPSG

We may prove theorem 7 using computation trees as follows.

Proof. To establish the result, we will hide a polynomial depth pruned OR computation tree in an polynomial depth branching RGPSG parse tree. The

major obstacle to our success is the RGPSG head feature convention, which ensures that all heads dominated by a common head will have the same head features. This means we cannot use the HFC to transfer unaltered tape squares from a configuration to its successors as we did for GPSG recognition.

The trick is that each RGPSG *category* will encode a polynomial depth pruned OR computation tree! Assume, without loss of generality, that our target computation tree has polynomial depth d and uses space s. We create s atomic features $g_1 \ldots g_s$ to encode the tape squares, as before, and d category-valued head features $f_1 \ldots f_d$ to represent a path in the unpruned OR computation tree, which is merely a pruned OR tree or a straight line. We will need the atomic features **state** and **head** to represent the machine state and head position, respectively.

We will use the ID rules to transfer the tape contents encoded in the 0-level categories from one embedded category to another. Thus, in the parse tree, embedded categories will obey the next-move relation. The head feature convention serves no useful role in this reduction–it merely ensures that all categories in the resultant parse tree are identical.

For each transition t found in δ we will create $s \cdot d(1+2(s-1))+1 = \theta(d \cdot s^2)$ ID rules as follows. Assume wolg that t changes a tape square from σ to σ' and state from q to q', and moves the tape head right. For each possible tape square i altered by t, $1 \leq i \leq s$, and each f_j other than f_d representing a configuration t could apply to, create the ID rule

$$\{[f_j\ \{[\texttt{state}\ q], [\texttt{head}\ i], [g_i\ \sigma]\}]\} \rightarrow \\ \{[f_{j+1}\ \{[\texttt{state}\ q'], [\texttt{head}\ i+1], [g_i\ \sigma']\}]\} : \text{`}ji|\text{'}$$

and the $2(s-1)$ ID rules for all tape squares k, $k \neq i$:

$$\{[f_j\ \{[\texttt{state}\ q], [\texttt{head}\ i], [g_k\ 1]\}]\} \rightarrow \\ \{[f_{j+1}\ \{[\texttt{state}\ q'], [\texttt{head}\ i+1], [g_k\ 1]\}]\} : \text{`}jk|\text{'}$$

$$\{[f_j\ \{[\texttt{state}\ q], [\texttt{head}\ i], [g_k\ 0]\}]\} \rightarrow \\ \{[f_{j+1}\ \{[\texttt{state}\ q'], [\texttt{head}\ i+1], [g_k\ 0]\}]\} : \text{`}jk|\text{'}$$

We need one more ID rule to terminate accepting configurations, which are in a special accept state q_a:

$$\{[f_d\ \{[\texttt{state}\ q_a]\}]\} \rightarrow \mathit{accept}$$

The distinguished start category will encode the root node of the computation tree embedded in the category-valued feature f_1.

Note that the parse tree successfully simulates the target computation tree iff it yields a string containing every possible pair ji for $1 \leq j < d$ and $1 \leq i \leq s$. The '$ji|$' substring in the terminal string indicates that any successful derivation of the terminal string has transferred the contents of the i^{th} tape square from the j^{th} configuration to it's successor (the $j+1^{th}$ configuration) in accordance with the next-move relation. These terminal strings look like '$11|12|13|\ldots|21|22|23|\ldots|ji|\ldots|(d-1)s|accept$'. □

C An RGPSG for English

Before presenting the English RGPSG in its entirety, we discuss some of the more tricky aspects of converting the FCRs and FSDs of the GKPS English grammar into RGPSG.

We must stop **+POSS** from getting forced on everything, perhaps with an SD **If true Then [POSS noBind]**. What does **+POSS** mean in GKPS anyway? Their analysis of **NP[+POSS]** and **PP[+POSS]** must be clarified.

We duplicate FSD 7: **[BAR 0]** $\supset \neg$**[VFORM PAS]**, which prevents random lexical categories from assuming a passive alternate, in a complicated and obtuse manner. We introduce a new head feature $\pm$**PAS** to indicate passive sentences and verb phrases, and no longer allow the **VFORM** feature to take **PAS** as a value. We also include four SDs to ensure that **VFORM** and **PAS** are mutually exclusive, and that **PAS** only appears in **[+V,-N]** categories. While this solution allows us to avoid both the implicit disjunctive consequence of FSD 7 and morass of problematic feature specifications, it is linguistically dubious. RGPSG (incorrectly) claims that human languages can have passive categories that are also finite, infinitival, and so on. On the other hand, this is just as offensive as the GPSG/RGPSG claim that some human languages can specify a category for the three features **VFORM**, **NFORM**, and **PFORM** simultaneously. The proper solution to this problem is to introduce finer internal structure in feature specifications. A more intricate version of the tree-like theory of features proposed in Gazdar and Pullum (1982) or the complex-symbol rules of Chomsky (1965) appear to be more linguistically and computationally desirable for these reasons.

FCR 10: **[+INV,BAR 2]** $\supset$ **[+SUBJ]** prevents **+INV** from "dripping through" the VP by the HFC. Note that **+INV** is only introduced on the mother *V2*[**+SUBJ**] category of lexical ID rules, and that the only instance of *V2*[**+SUBJ**] as the head daughter of a (potential) *VP* is in the ID rule 35

$$V2 \rightarrow X2 : X2[\texttt{+ADV}] \tag{35}$$

We include the simple default SD: If **[-SUBJ]** Then **[-INV]** to prevent **+INV** from rising through any daughter *VP*'s. Alternately, we could replace 35 with 36, which could be linguistically incorrect.

$$V2[\texttt{-INV}] \rightarrow X2 : X2[\texttt{+ADV}] \tag{36}$$

Lastly, GKPS fn.2 on page 73 says that the category *S*[VFORM PAS] is invalid in English, yet GKPS fail to enforce this constraint in their GPSG for English. If it is actually desirable to rule out the suspect category, then include SD: If [+SUBJ] Then [-PAS].

Without further ado, the following RGPSG is the result of translating the GKPS grammar for English into the RGPSG formal system.

```
;-----------------SYNTACTIC FEATURES-----------------

CASE     {ACC,NOM}
COMP     {for,that,whether,if,noBind}
CONJ     {and,both,but,neither,either,nor,or}
GER      {+,-}
NEG      {+,-}
NULL     {+,-}
POSS     {RECP,REFL}
REMOR    {RECP,REFL}
WHMOR    {R,Q,FR,EX}

; HEAD features

AGR      {}
ADV      {+,-}
AUX      {+,-}
INV      {+,-}
LOC      {+,-}
N        {+,-}
NFORM    {there,it,NORM}
PAS      {+,-}
PAST     {+,-}
PER      {1,2,3}
PFORM    {to,by,for,about,of,with,....}
PLU      {+,-}
PRD      {+,-}
V        {+,-}
VFORM    {BSE,FIN,INF,PRP,PSP}

; BHEAD features

BAR      {0,1,2,noBind}
SUBCAT   {1,...,48,for,that,whether,if,
```

```
          and,both,either,neither,but,nor,or,not}
SUBJ    {+,-}

; FOOT features

RE      {}
SLASH   {}
WH      {}


;-----------------ABBREVIATIONS-----------------------

S ::= [+V,-N,+SUBJ,BAR 2]
VP ::= [+V,-N,-SUBJ,BAR 2]
NP ::= [-V,+N,BAR 2]
AP ::= [+V,+N,BAR 2]
PP ::= [-V,-N,BAR 2]
V ::= [+V,-N]
N ::= [+N,-V]
A ::= [+N,+V]
P ::= [-N,-V]

+it ::= [AGR NP[NFORM it]]
+there ::= [AGR NP[NFORM there]]
+NORM ::= [AGR NP[NFORM NORM]]
+Q ::= [WH NP[WHMOR Q]]
+R ::= [WH NP[WHMOR R]]
Deg ::= {[SUBCAT 23,BAR noBind]}
~F ::= [F noBind]


;----------------SIMPLE DEFAULTS-------------------------

If [SUBCAT] Then [BAR 0]
If [SUBCAT] Then [SLASH noBind]
If (~[+PAS] & ~[PRP] & [+V,-N] Then [PRD noBind]
If [+SUBJ,WH] Then [COMP noBind]
If [+SUBJ,INF] Then [COMP for]
If [+V,-N,-SUBJ] Then [AGR NP[NFORM NORM]]
If [SLASH] Then [WH noBind]
If [WH] Then [SLASH noBind]
If A1 Then [WH noBind]
If VP then [WH noBind]
If true Then [NULL noBind]
```

```
If true Then [CONJ noBind]
If [-SUBJ] Then [-INV]
If [VFORM] Then [-PAS]
If [+PAS] Then [VFORM noBind]
If [SUBCAT] & [+V,-N] Then [-PAS]
If [-V] | [+N] Then [PAS noBind]

; to duplicate FCR 5, which appears to be useless . . . .
If [+SUBJ] | (~[FIN] & [VFORM]) Then [PAST noBind]

; these four are all questionable
If true Then [-INV]  ; means extraposed S's must be matrix S's
If true Then [CASE NOM] ;BUT no case defaults should be allowed
If [+N,-V,BAR 2] Then [CASE ACC] ; no case defaults should be allowed
If [+SUBJ] Then [-PAS] ; GKPS p.73 fn2 implies this is needed

;----------------ID RULES----------------------------

VP -> [1] :                      %(die eat sing run succeed weep occur
   dine elapse grow look)
VP -> [2] : NP                   %(sing love close prove succeed
   abandon enlighten castigate slap eat devour grow bring trade)
VP -> [3] : NP, PP[to]           %(give sing throw hand trade)
VP -> [4] : NP, PP[for]          %(buy cook reserve save trade)
VP -> [5] : NP, NP               %(spare hand give buy trade)
VP -> [6] : NP, PP[+LOC]         %(put place stand)
VP -> [7] : X2[+PRD]             %(be)
; Sells p.130 claims this should be VP[+AUX] -> [7] : X2[+PRD]
VP -> [8] : NP, S[FIN]           %(persuade convince tell)
VP -> [9] : (PP[to]), S[FIN]     %(concede admit)
VP -> [10] : S[BSE]              %(prefer desire insist)
VP -> [11] : (PP[of]), S[BSE]    %(require)
VP[INF,+AUX,AGR NP] -> [12] : VP[BSE]      %(to)
VP -> [13] : VP[INF]             %(continue tend seem want)
VP -> [14] : V2[INF,+NORM]       %(prefer intend)
VP -> [15] : VP[INF,+NORM]       %(try attempt want)
VP -> [16] : (PP[to]), VP[INF]          %(seem appear)
VP -> [17] : NP, VP[INF]                %(believe expect)
VP -> [18] : NP, VP[INF,+NORM]          %(persuade force)
VP -> [19] : (NP), VP[INF,+NORM]        %(promise)
VP[AGR S] -> [20] : NP                  %(bother amuse)
VP[+it] -> [21] : (PP[to]), S[FIN]      %(seem appear)
VP[AGR NP[there,PLU_1]] -> [22] : NP[PLU_1]      %(be)
VP -> [40] : S[FIN]                     %(believe say regret)
```

```
VP -> [43] : S[+Q]                          %(wonder ask inquire)
VP[+it] -> [44] : NP, S[+R]                 %(be)
VP[+it] -> [44] : X2, S[FIN]/X2             %(be)
VP -> [45] : PP[of]                         %(approve)
VP -> [48], X2[-SUBJ,CONJ and] :            %(come go)
VP -> X2[-SUBJ] : AP[+ADV]                  %    ;is this too constrained?
VP -> X2[-SUBJ] : X2[+ADV]                  %    ;adverbial adjuncts to VP
S -> X2[+SUBJ] : X2[+ADV]                   %    ;adverbial adjuncts to S
V0[+NEG] -> X0[+AUX] : [SUBCAT not,'BAR]    %   ; "was not"


A2 -> X1 : (Deg)                %
AP[-ADV] -> X1 : (AP[+ADV])     %
A1 -> [24] : PP[about]          %(angry glad curious)
A1[AGR S] -> [25] : PP[to]      %(apparent obvious certain)
A1 -> [26] : S[FIN]             %(afraid aware amazed)
A1 -> [27] : S[BSE]             %(insistent adamant determined)
A1 -> [28] : VP[INF]            %(likely certain sure)
A1 -> [29] : V2[INF,+NORM]      %(anxious eager)
A1 -> [42] : V2[INF]/NP[-NOM]   %(easy)

;Spec ::= determiners, possessive phrases, limited set of quantifying
;         APs (e.g. many, few)
; N1 -> X1 : Modifier   by Andrews(1983)

NP -> X1 : Spec              %
NP[+SUBJ] -> X1 : NP[+POSS]  %      ; X1 prevents multi-DET or -gerunds
          ; [+SUBJ] says i have a subject, allowing refl/recip
          ; to work properly. Multiple-possesives must be allowed.
NP[+POSS] -> X1  : "'s"      %
N1 -> X1 : PP[+POSS]         %
N1 -> X1 : PP                %
N1 -> X1 : S[+R]             %
N1 -> X1 : N0                %  ;for noun-noun modification
N1 -> [30] :                      %(death disappearance laughter)
N1 -> [31] : PP[with], PP[about]  %(argument consultation conversation)
N1 -> [32] : S[COMP that]         %(belief implication proof notion idea)
N1 -> [33] : S[BSE,COMP that]     %(request insistence proposal)
N1 -> [34] : V2[INF]          %(plan wish desire)
N1 -> [35] : PP[of]           %(king sister inside love seduction criticism)
N1 -> [36] : PP[of], PP[to]   %(gift announcement surrender)
N1 -> [37] : PP[of,+GER]      %(dislike admission memory habit prospect idea)

; depending on how LP statements are enforced, N-N mod rule might have to be
```

```
; N1 -> X1 : N0[SUBCAT {30,31,32,33,34,35,36,37}], i.e. heinous.
; because LP says SUBCAT always comes first.


;Mod ::= almost, totally, immediately, right, three feet, nearly
;Mod bears the SUBCAT feature and therefore precedes X1 in the first PP rule.
; maybe P1[+POSS] should be PP[+POSS] or the grammar crashes

PP -> X1 : Mod                    %
P1 -> [38] : NP                   %(to underneath in beside)
P1 -> [39] : PP[of]               %(out forward in_front in_back)
P1[+POSS] -> [41] : NP[+POSS]     %(of)


S[COMP noBind] -> X2[-SUBJ,AGR X2] : X2                 %
S[COMP noBind] -> X2[+SUBJ]/X2 : X2                     %
S[COMP that,FIN] -> S[COMP noBind] : [SUBCAT that]      %
S[COMP that,BSE] -> S[COMP noBind] : [SUBCAT that]      %
S[COMP whether] -> S[COMP noBind] : [SUBCAT whether]    %
S[COMP if] -> S[COMP noBind] : [SUBCAT if]              %
S[COMP for,INF] -> S[COMP noBind] : [SUBCAT for]        %


; iterating coordination schema, * means positive transitive closure +
X -> [CONJ and], X* :                   %
X -> [CONJ noBind], [CONJ and]* :       %
X -> [CONJ neither], [CONJ nor]* :      %
X -> [CONJ or], X* :                    %
X -> X, [CONJ or]* :                    %

; binary coordination schema
X -> [CONJ both], [CONJ and] :          %
X -> [CONJ either], [CONJ or] :         %
X -> X, [CONJ but] :                    %


;--------------------LP STATEMENTS-----------------------

[SUBCAT] << [SUBCAT {unbound,noBind}]
[+N] << P2 << V2
[CONJ {both,either,neither,noBind}] << [CONJ {and,but,nor,or}]


;--------------------METARULES---------------------------
```

```
; passive metarule
!VP -> W, NP! => !VP[+PAS] -> W, (PP[by])!

; subject-aux inversion metarule
!V2[-SUBJ] -> W! => !V2[+INV,+AUX,+SUBJ,FIN] -> W, NP!

; extraposition metarule
!X2[AGR S] -> W! => !X2[+it] -> W, S!

;complement omission metarule
![+N,BAR 1] -> W! => ![+N,BAR 1] -> !

; slash termination metarule 1
!X -> W, X2! => !X -> W, X2[+NULL]!

; slash termination metarule 2
!X -> W, V2[+SUBJ,FIN]! => !X/NP -> W, V2[-SUBJ]!

;---------------------SOME LEXICAL RULES-------------------------

; universal lexical rule,
; where two X2's are linked for all features but NULL
<"", X2[+NULL]_1/X2_1>

<"quickly", A[+ADV]>
<"excessively", A[+ADV]>
<"aren't", V[+AUX,+NEG,PER 1,-PLU,+INV]>

<"that", [SUBCAT that,'BAR]>
<"whether", [SUBCAT whether,'BAR]>
<"if", [SUBCAT if,'BAR]>
<"for", [SUBCAT for,'BAR]>
<"both", [SUBCAT both,'BAR]>
<"either", [SUBCAT either,'BAR]>
<"neither", [SUBCAT neither,'BAR]
<"and", [SUBCAT and,'BAR]>
<"but", [SUBCAT but,'BAR]>
<"nor", [SUBCAT nor,'BAR]>
<"or", [SUBCAT or,'BAR]>

<"it", NP[PRO,-PLU,NFORM it]>
<"there", NP[PRO,NFORM there]>
```

```
;lexical rules to discharge PFORM
<"of", PO[PFORM of]>
<"to", PO[PFORM to]>
<"with", PO[PFORM with]>
<"about", PO[PFORM about]>
<"by", PO[PFORM by]>
<"for", PO[PFORM for]>

<"what", NP[+Q]>
<"which", NP[+Q]>
<"which", NP[+R]>
<"which", Det[+Q]>
<"which", Det[+R]>
<"whose", Det[+POSS,+Q]>
<"whose", Det[+POSS,+R]>

<"so", Deg>
<"too", Deg>
<"very", Deg>
```

D Syntactic features in GPSG and RGPSG

These notes contain an informal description of the GPSG/RGPSG feature system: what each feature means, and how it is used. A typical entry is of the form:

```
#<feature>      {<permissible-feature-values>}

[<feature-specification-1>]: what <feature-specification-1> means.
[<feature-specification-2>]: what <feature-specification-2> means.
```

```
#BAR     {0,1,2,noBind}

[BAR 0]: for pure lexical entries, i.e. words.
(nouns, verbs, prepositions, adjectives, adverbs). Almost
```

all preterminals will have the [BAR 0] feature.

[BAR 1]: for lexical entries and their complements
(e.g. for '[man [that I knew]]' because it is missing its specifier)

[BAR 2]: for complete phrases (NP, VP, PP, AdjP), including both complements and specifiers. The only [BAR 2] lexical categories are for "there" and "it".

#CASE {NOM,OBL,OBJ} ; this feature is in BHEAD.

[CASE NOM]: nominative case is assigned by a VP to its subject
"I", "who", "he" are all NP[CASE NOM].

[CASE OBJ]: objective case is assigned by a verb to its NP complements
"him", "me" are NP[CASE OBJ,OBL]. The GPSG specification [CASE ACC]
(accusative case) includes both the objective and oblique cases of
RGPSG.

[CASE OBL]: oblique case is assigned by a preposition to its NP.
"whom" is NP[CASE OBL], although this is a weak distinction.

#COMP {for,that,whether,if,noBind}

This feature labels a clause with the lexical item appearing in its complementizer position. The value of COMP is morphologically realized. For example, "whether John is a fool" is S[COMP whether].

#CONJ {and,both,but,neither,either,nor,or,noBind}

This feature labels a conjunct with the conjunction word that is associated with it. For example, in the coordinate structure "Either Bill or Bob died", the first conjunct would be NP[CONJ either], and the second conjunct would be NP[CONJ or].

#GENDER {M,F,N} "gender"

[GENDER M]: for masculine gender
[GENDER F]: for feminine gender
[GENDER N]: for neuter gender

#GER {+,-} "gerund"

[GER +]: for gerunds (verbs functioning as nouns with an "-ing" ending)

[GER -]: for nouns that may not be gerunds.

#NEG {+,-} "negation"

[NEG +]: for all negated categories ("not") and categories in the scope of negation. Contractions ("wasn't") should be separated into "was not", and "not" should be replaced by the [NEG +] preterminal.

[NEG -]: for elements that may not be negated. No lexical entry is [NEG -].

#NULL {+,-}

[NULL +]: for phonologically empty elements, such as traces and gaps.
[NULL -]: for phonologically overt elements. All lexical entries are [NULL -].

#PER {1,2,3} "person"

[PER 1]: for first person nouns "I", "me"
[PER 2]: for second person nouns "you"
[PER 3]: for third person nouns "he", "she", "it"

#POSS {+,-}

[POSS +]: for possessives, i.e. nouns with genetive case such as "his".

#SUBCAT {1...48,for,that,whether,if,
and,both,either,neither,but,nor,or,not} "subcategorization"

All words have at least one value for their SUBCAT feature. In the feature specification [SUBCAT n], n is a subcategorization index.

[SUBCAT 1]: intransitive verbs (die, eat, sing, run)
[SUBCAT 2]: verbs appearing in the VP [V NP] (e.g., SING a song)
[SUBCAT 3]: verbs appearing in the VP [V NP to NP] (GIVE John a book)
[SUBCAT 4]: [V NP for NP] verbs (BUY a book for John)
[SUBCAT 5]: [V NP NP] verbs (GIVE Bill the book)
[SUBCAT 6]: [V NP PP[+LOC]] verbs (PUT the disk in the water)

The meaning of numerical SUBCAT values is determined by the ID rules they appear in; see the GKPS or RGPSG grammars for English. The nonnumerical SUBCAT values (for example, "for" or "that") represent themselves: the terminal "for" is associated with the [SUBCAT for]

```
feature specification, "that" is associated with [SUBCAT that], and so
on.  Note that there are (at least) two lexical entries for "for":
<"for", [SUBCAT for,BAR noBind]>
<"for", [-N,-V,BAR 0,PFORM for]>

#SUBJ    {+,-} "subject"

[SUBJ +]: for categories with subjects (clauses, for example).
[SUBJ -]: for categories missing a subject (verb phrases, for example).

#REMOR   {RECP,REFL}   "REciprocal/REflexive morphology"

[REMOR RECP]: for reciprocals ("themselves").
[REMOR REFL]: for reflexives ("each other").

#WHMOR   {R,Q,FR,EX} "WH- morphology"

For wh- nouns, pronouns, and phrases, e.g. what, who, whom, where.
The entries for these words are highly ideosyncratic: see the lexicon
fragment in the RGPSG for English given below.

[WHMOR R]: for relative wh- pronouns (whom, which, whose) in relative clauses.
[WHMOR Q]: for interrogative pronouns (what, which, whose) in questions.
[WHMOR FR]: ?
[WHMOR EX]: ?

#AGR     {}                ; {} indicates category-valued feature

AGR appears only on verbs in English. Its value is the type of subject
that the verb selects. "frighten" can only take -ABSTRACT noun phrase
subjects, so its lexical entry might be
<"frighten", V0[SUBCAT 2, AGR NP[+PLU,-ABSTRACT]]>

#ADV     {+,-} "adverbial"

[ADV +]: for adverbial adjectives (i.e. adverbs).
[ADV -]: for nonadverbial adjectives.

#AUX     {+,-} "auxiliary"

[AUX +]: for auxiliary verbs ("is").
[AUX -]: for verbs that are not auxiliaries.

#INV     {+,-} "invertable"
```

```
[INV +]: for invertable verbs, typically the finite auxiliaries. Items
that must invert (first-singular "aren't", "need") are labeled [INV +].
Examples: Aren't you cold? Need we die?
[INV unBound]: for verbs that optionally invert (some finite auxiliaries).
[INV -]: for everything else, e.g. for the auxiliary "better" because
it can't invert.

#LOC      {+,-} "locative"

[LOC +]: for all categories that are locations (water, Egypt, in the house).
[LOC -]: for categories that aren't locations (John, virtue).

#N        {+,-} "nominal"

[N +]: for nouns and adjectives/adverbs.
[N -]: for verbs and prepositions.

#NFORM    {there,it,NORM}

[NFORM there]: only for the pleonastic noun phrase "there"
<"there", NP[PRO,NFORM there]>

[NFORM it]: only for the pleonastic noun phrase "it"
<"it", NP[PRO,-PLU,NFORM it]>

[NFORM NORM]: for all other nouns. Note that "it" may appear
in a non-pleonastic reading as a pronoun.

#PAS      {+,-}  "passive"

[PAS +]: for verbs with passive morphology (given, kissed, known, believed).
[PAS -]: for nonpassive verbs.

#PAST     {+,-}  "past tense"

[PAST +]: for past tense verbs (gave, knew).
[PAST -]: for present tense verbs (give, know).

#PFORM    {to,by,for,about,of,with,. . .}

All prepositions will have a PFORM specification whose value is
the preposition itself, e.g. <"about", PO[PFORM about]>.
```

```
#PLU      {+,-} "plural"

[PLU +]: for plural categories (e.g. plural nouns and verbs).
[PLU -]: for singular categories.

#PRD      {+,-} "predicate"

[PRD +]: for predicates.  This includes adjectives ("happy") and
predicate nominals (anything that denotes a set of things, e.g.
"doctor" because it denotes the set of all doctors).

[PRD -]: for words that cannot be predicates (e.g. determiners).

#V        {+,-} "verbal"

[V +]: for verbs and adjectives/adverbs.
[V -]: for nouns and prepositions.

#VFORM    {BSE,FIN,INF,PRP,PSP}

This feature labels a verb and its projections (V0, V1, VP, and S)
with the morphological class of the verb. In GPSG, [VFORM PAS] labels
passive verbs and their projections. In RGPSG, [PAS +] performs that
function.

[VFORM BSE]: uninflected, untensed base verb form
[VFORM FIN]: inflected, tensed finite verb form
[VFORM INF]: infinitival verb form. Signifies inflection w/o AGR plus V0,
             e.g. "to be" or "to V0".
[VFORM PRP]: purposive verb form, e.g. "cleaning a bone". (not implemented)
[VFORM PSP]: past participle. (not implemented)
```

Contents

CS-TR Scanning Project
Document Control Form

Date : 6/29/95

Report # AI-TR 1700

Each of the following should be identified by a checkmark:
Originating Department:

☒ Artificial Intellegence Laboratory (AI)
☐ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 98 (106-IMAGES)
Not to include DOD forms, printer instructions, etc... original pages only.

Originals are:
☒ Single-sided or
☐ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☒ Laser Print
☐ InkJet Printer ☐ Unknown ☐ Other:_____

Check each if included with document:

☒ DOD Form (2) ☐ Funding Agent Form ☒ Cover Page
☒ Spine ☐ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number):_____

Photographs/Tonal Material (by page number):_____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1) UN#'ED TITLE PAGE	
(2-98) PAGES #'ED 1-97	
(99-103) SCANCONTROL, COVER, SPINE, DOD(2)	
(104-106) TRGT'S (3)	

Scanning Agent Signoff:
Date Received: 6/29/95 Date Scanned: 7/13/95 Date Returned: 7/13/95

Scanning Agent Signature: Michael W. Cook

REPORT DOCUMENTATION PAGE

1. REPORT NUMBER: AI-TR 1170	2. GOVT ACCESSION NO.
3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle): Computational Structure of GPSG Models: Revised generalized phrase structure grammar	5. TYPE OF REPORT & PERIOD COVERED: technical report
6. PERFORMING ORG. REPORT NUMBER	
7. AUTHOR(s): Eric Sven Ristad	8. CONTRACT OR GRANT NUMBER(s): N00014-80-C-0505
9. PERFORMING ORGANIZATION NAME AND ADDRESS: Artificial Intelligence Laboratory, 545 Technology Square, Cambridge, MA 02139	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS: Advanced Research Projects Agency, 1400 Wilson Blvd., Arlington, VA 22209	12. REPORT DATE: September 1989 13. NUMBER OF PAGES: 93
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office): Office of Naval Research, Information Systems, Arlington, VA 22217	15. SECURITY CLASS. (of this report): UNCLASSIFIED 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Distribution is unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

None

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

linguistics
complexity
GPSG
natural language
computational structure
computational complexity

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See reverse

Block 20 cont'd

A central goal of mathematical linguistics is to precisely determine the power of a linguistic theory. Traditionally, formal language theory (the Chomsky hierarchy) and its generative power analyses have translated this question into the narrower question of how unrestricted the rule format of a theory is. Modern computational complexity theory offers another, more useful, translation: how much of what computational resources does a theory consume? Complexity theory also offers a new perspective on descriptive adequacy. In a descriptively adequate linguistic theory, the structural descriptions *and* computational power of the theory match those of an ideal speaker-hearer.

The primary goal of this paper is to demonstrate how considerations from computational complexity theory can inform grammatical theorizing. To this end, the paper revises generalized phrase structure grammar (GPSG) linguistic theory so that its computational power more closely matches the limited computational ability of an ideal speaker–hearer. A second goal is to provide a theoretical framework within which to better understand the wide range of GPSG models that have appeared in the theoretical and computational linguistics literature, embodied in formal definitions as well as in implemented computer programs.

The paper begins with an outline and intuitive complexity analysis of the GPSG formal system of Gazdar, Klein, Pullum, and Sag (1985). Subsequently, revisions to the formal system are motivated by complexity and generative concerns. The revised system is presented along with an account of topicalization, expletive pronouns, and parasitic gaps. This work falls within the GPSG approach to linguistics. Revised GPSG is, however, less opaque, more tractable, and more linguistically constrained than standard GPSG theory: GPSG Recognition is EXP-POLY time hard, while RGPSG Recognition is NP-complete.

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**